VeriLSM

Contents

1 Element

theory *Element*
  imports
    Main
    HOL.Real
    HOL−Word.Word-Bitwise
begin

In this theory, we introduce the elementary datatype and data structure of Kernel

1.1 uidgid

type-synonym *k--kernel-uid32-t* = *nat*
type-synonym *k--kernel-gid32-t* = *nat*
type-synonym *uid-t* = *k--kernel-uid32-t*
type-synonym *gid-t* =*k--kernel-gid32-t*
typedecl *uid-gid-map*
record *kuid-t* = *uval* :: *uid-t*

record *kgid-t* = *gval* :: *gid-t*

type-synonym *usnum* = *nat*

record *user-namespace* = *uid-map* :: *uid-gid-map*
  gid-map :: *uid-gid-map*
  projid-map :: *uid-gid-map*
  count :: *int*
  ns-level :: *int*
  owner :: *kuid-t*
  group :: *kgid-t*
  u-flags :: *nat*
  ns-parent :: *usnum*

type-synonym *ns* = *user-namespace*

definition *DEFAULT-OVERFLOWUID* ≡ *65534*
definition *DEFAULT-OVERFLOWGID* ≡ *65534*

definition *overflowuid* ≡ *DEFAULT-OVERFLOWUID*
definition *overflowgid* ≡*DEFAULT-OVERFLOWGID*

consts *CONFIG-MULTIUSER* :: *bool*

definition *k--kuid-val* :: *kuid-t* ⇒ *uid-t*
  where *k--kuid-val uid′* ≡ *if CONFIG-MULTIUSER then* (*uval uid′*) *else 0*

definition *k--kgid-val* :: *kgid-t* ⇒ *gid-t*
  where *k--kgid-val gid′* ≡ *if CONFIG-MULTIUSER then* (*gval gid′*) *else 0*

definition *KUIDT-INIT value* ≡ (|*uval* = *value* |)

definition *KGIDT-INIT value* ≡ (|*gval* = *value* |)

definition *make-kuid* :: *user-namespace*⇒ *uid-t*⇒ *kuid-t*
  where *make-kuid from uid′* ≡ *KUIDT-INIT uid′*

definition *make-kgid* :: *user-namespace*⇒ *gid-t*⇒ *kgid-t*
  where *make-kgid from gid′* ≡ *KGIDT-INIT gid′*

definition *from-kuid* :: *ns* ⇒ *kuid-t* ⇒ *uid-t*
  where *from-kuid to kuid* ≡ *k--kuid-val kuid*

definition *from-kgid* :: *ns* ⇒ *kgid-t* ⇒ *gid-t*
  where *from-kgid to kgid* ≡ *k--kgid-val kgid*

definition *from-kuid-munged* :: *ns* ⇒ *kuid-t* ⇒ *uid-t*
  where *from-kuid-munged to kuid* ≡
    let uid = *from-kuid to kuid*
    in
    if uid = *65535* − *1 then overflowuid*
    else uid

definition *from-kgid-munged* :: *ns* ⇒ *kgid-t* ⇒ *gid-t*
  where *from-kgid-munged to kgid* ≡
    let gid = *from-kgid to kgid*
    in
    if gid = *65535* − *1 then overflowgid*
    else gid

definition *kuid-has-mapping* :: *ns* ⇒ *kuid-t* ⇒ *bool*
where *kuid-has-mapping ns uid* ≡ *True*

definition *kgid-has-mapping* :: *ns* ⇒ *kgid-t* ⇒ *bool*
where *kgid-has-mapping ns gid* ≡ *True*

definition *uid-eq* :: *kuid-t* ⇒ *kuid-t* ⇒ *bool*
where *uid-eq left right* ≡ *k--kuid-val left* = *k--kuid-val right*

1.2 stat$_h$

definition *S-IFMT* ≡ *00170000*
definition *S-IFSOCK* ≡ *0140000*
definition *S-IFLNK* ≡ *0120000*
definition *S-IFREG* ≡ *0100000*
definition *S-IFBLK* ≡ *0060000*
definition *S-IFDIR* ≡ *0040000*
definition *S-IFCHR* ≡ *0020000*
definition *S-IFIFO* ≡ *0010000*
definition *S-ISUID* ≡ *0004000*
definition *S-ISGID* ≡ *0002000*
definition *S-ISVTX* ≡ *0001000*

definition *S-ISLNK m* ≡ (((*m*) *AND S-IFMT*) = *S-IFLNK*)
definition *S-ISREG m* ≡ (((*m*) *AND S-IFMT*) = *S-IFREG*)
definition *S-ISDIR m* ≡ (((*m*) *AND S-IFMT*) = *S-IFDIR*)
definition *S-ISCHR m* ≡ (((*m*) *AND S-IFMT*) = *S-IFCHR*)
definition *S-ISBLK m* ≡ (((*m*) *AND S-IFMT*) = *S-IFBLK*)
definition *S-ISFIFO m* ≡ (((*m*) *AND S-IFMT*) = *S-IFIFO*)
definition *S-ISSOCK m* ≡ (((*m*) *AND S-IFMT*) = *S-IFSOCK*)

definition *S-IRWXU* ≡ *00700*
definition *S-IRUSR* ≡ *00400*
definition *S-IWUSR* ≡ *00200*
definition *S-IXUSR* ≡ *00100*

definition *S-IRWXG* ≡ *00070*
definition *S-IRGRP* ≡ *00040*
definition *S-IWGRP* ≡ *00020*
definition *S-IXGRP* ≡ *00010*

definition *S-IRWXO* ≡ *00007*
definition *S-IROTH* ≡ *00004*
definition *S-IWOTH* ≡ *00002*
definition *S-IXOTH* ≡ *00001*

definition *S-IRWXUGO* ≡ *bitOR* (*bitOR S-IRWXU S-IRWXG*) *S-IRWXO*

definition *S-IALLUGO* ≡ *bitOR*(*bitOR* (*bitOR* *S-ISUID* *S-ISGID*) *S-ISVTX*) *S-IRWXUGO*

definition *S-IRUGO* ≡ *bitOR* (*bitOR* *S-IRUSR* *S-IRGRP*) *S-IROTH*
definition *S-IWUGO* ≡ *bitOR*(*bitOR* *S-IWUSR* *S-IWGRP*) *S-IWOTH*
definition *S-IXUGO* ≡ *bitOR* (*bitOR* *S-IXUSR* *S-IXGRP*) *S-IXOTH*

definition *STATX-TYPE* ≡ *0x00000001*
definition *STATX-MODE* ≡ *0x00000002*
definition *STATX-NLINK* ≡ *0x00000004*
definition *STATX-UID* ≡ *0x00000008*
definition *STATX-GID* ≡ *0x00000010*
definition *STATX-ATIME* ≡ *0x00000020*
definition *STATX-MTIME* ≡ *0x00000040*
definition *STATX-CTIME* ≡ *0x00000080*
definition *STATX-INO* ≡ *0x00000100*
definition *STATX-SIZE* ≡ *0x00000200*
definition *STATX-BLOCKS* ≡ *0x00000400*
definition *STATX-BASIC-STATS* ≡ *0x000007ff*
definition *STATX-BTIME* ≡ *0x00000800*
definition *STATX-ALL* ≡ *0x00000fff*
definition *STATX--RESERVED* ≡ *0x80000000*

definition *STATX-ATTR-COMPRESSED* ≡ *0x00000004*
definition *STATX-ATTR-IMMUTABLE* ≡ *0x00000010*
definition *STATX-ATTR-APPEND* ≡ *0x00000020*
definition *STATX-ATTR-NODUMP* ≡ *0x00000040*
definition *STATX-ATTR-ENCRYPTED* ≡ *0x00000800*

definition *STATX-ATTR-AUTOMOUNT* ≡ *0x00001000*

1.3 cred

type-synonym *kuid* = *nat*
type-synonym *kgid* = *nat*
record *kernel-cap-struct* = *kcap* :: *int list*

type-synonym *kernel-cap-t* = *kernel-cap-struct*

record *Cred* = *uid* :: *kuid-t*
gid :: *kgid-t*
suid :: *kgid-t*
sgid :: *kgid*
euid :: *kuid-t*
egid :: *kgid-t*
fsuid :: *kuid-t*
fsgid :: *kgid-t*
user-ns :: *ns*

cap-effective :: *kernel-cap-t*

1.4 $\mathbf{fs}_h$

definition *ATTR-MODE* ≡ (*1* << *0*)
definition *ATTR-UID* ≡ (*1* << *1*)
definition *ATTR-GID* ≡ (*1* << *2*)
definition *ATTR-SIZE* ≡ (*1* << *3*)
definition *ATTR-ATIME* ≡ (*1* << *4*)
definition *ATTR-MTIME* ≡ (*1* << *5*)
definition *ATTR-CTIME* ≡ (*1* << *6*)
definition *ATTR-ATIME-SET* ≡ (*1* << *7*)
definition *ATTR-MTIME-SET* ≡ (*1* << *8*)
definition *ATTR-FORCE* ≡ (*1* << *9*)
definition *ATTR-KILL-SUID* ≡ (*1* << *11*)
definition *ATTR-KILL-SGID* ≡ (*1* << *12*)
definition *ATTR-FILE* ≡ (*1* << *13*)
definition *ATTR-KILL-PRIV* ≡ (*1* << *14*)
definition *ATTR-OPEN* ≡ (*1* << *15*)
definition *ATTR-TIMES-SET* ≡ (*1* << *16*)
definition *ATTR-TOUCH* ≡ (*1* << *17*)

type-synonym *dname* = *string*
record *ovl-fs* = *creator-cred* :: *Cred*

record *super-block* = *s-magic* :: *nat*
s-id :: *string*
s-root :: *dname*
s-user-ns :: *user-namespace*
s-fs-info :: *ovl-fs*
s-flags :: *nat*
s-iflags :: *nat*

record *file-lock* = *fl-flags* :: *nat*
fl-type :: *char*

definition *SB-I-NOEXEC* ≡ *0x00000002*
definition *O-ACCMODE* ≡ *00000003*
definition *FMODE-NONOTIFY* ≡ *0x4000000*
definition *OPEN-FMODE flag* ≡ *bitOR* (*flag* + *1* *AND* *O-ACCMODE*) (*flag* *AND* *FMODE-NONOTIFY*)

1.5 $\mathbf{fat}_h$

definition *VFAT-SFN-DISPLAY-LOWER* ≡ *0x0001*
definition *VFAT-SFN-DISPLAY-WIN95* ≡ *0x0002*
definition *VFAT-SFN-DISPLAY-WINNT* ≡ *0x0004*
definition *VFAT-SFN-CREATE-WIN95* ≡ *0x0100*
definition *VFAT-SFN-CREATE-WINNT* ≡ *0x0200*

definition *FAT-ERRORS-CONT* $\equiv$ *1*
definition *FAT-ERRORS-PANIC* $\equiv$ *2*
definition *FAT-ERRORS-RO* $\equiv$ *3*

definition *FAT-NFS-STALE-RW* $\equiv$ *1*
definition *FAT-NFS-NOSTALE-RO* $\equiv$ *2*

type-synonym *umode-t* = *nat*
record *fat-mount-options* = *fs-uid* :: *kuid-t*
fs-gid :: *kgid-t*
fs-fmask :: *nat*
fs-dmask :: *nat*
rodir :: *nat*

record *msdos-sb-info* = *sbi-options* :: *fat-mount-options*

definition *ATTR-RO* $\equiv$ *1*
definition *ATTR-DIR* $\equiv$ *16*
type-synonym *u8* = *char*

definition *fat-make-mode* :: *msdos-sb-info* $\Rightarrow$ *u8* $\Rightarrow$ *umode-t* $\Rightarrow$ *umode-t*
where *fat-make-mode sbi attrs m* $\equiv$
let
attrs' = *int*(*of-char attrs*);
mode = *if* ((*attrs' AND ATTR-RO*) $\neq$ *0*) $\wedge$ $\neg$(((*attrs' AND ATTR-DIR*) $\neq$ *0*)$\wedge$ $\neg$(*rodir*(*sbi-options sbi*)) $\neq$ *0*)
then ((*int m*) *AND* (*NOT S-IWUGO*))
else (*int m*);
ret = (*if* (*attrs' AND ATTR-DIR*) $\neq$ *0 then*
bitOR (*mode AND* (*NOT* (*int*(*fs-dmask*(*sbi-options sbi*))))) *S-IFDIR*
else
bitOR (*mode AND* (*NOT* (*int*(*fs-fmask*(*sbi-options sbi*))))) *S-IFREG*)
in (*nat ret*)

1.6 other kernel data structures

type-synonym *mode* = *int*
type-synonym *pages* = *int*
type-synonym *flags* = *nat*
type-synonym *kernel-ulong-t* = *nat*
typedecl *spinlock-t*
typedecl *security*
type-synonym *loff-t* = *int*
typedecl *fown-struct*
typedecl *kernel-load-data-id*
typedecl *kernel-read-file-id*

type-synonym *pid-t* = *int*
typedecl *siginfo*
typedecl *file-system-type*

type-synonym *process-id* = *nat*
type-synonym *inum* = *nat*
type-synonym *t-sb* = *nat*
type-synonym *fname* = *string*
type-synonym *pname* = *string*
type-synonym *msg-qid* = *int*
type-synonym *msg-mid* = *int*
type-synonym *keyid* = *int*
type-synonym *socketdesp* = *int*

record *vm-area-struct* = *vm-end* :: *nat*
vm-start :: *nat*
vm-flags :: *nat*

record *mm-struct* = *mmap* :: *vm-area-struct*

record *msg-msg* = *m-type* :: *int*
m-ts :: *nat*

datatype *Process-flags* = *PF-IDLE* | *PF-EXITING* | *PF-EXITPIDONE* | *PF-VCPU*| *PF-WQ-WORKER*
| *PF-FORKNOEXEC* | *PF-MCE-PROCESS* | *PF-SUPERPRIV*| *PF-DUMPCORE*

| *PF-SIGNALED* | *PF-MEMALLOC* | *PF-NPROC-EXCEEDED* | *PF-USED-MATH*

| *PF-USED-ASYNC* | *PF-NOFREEZE* | *PF-FROZEN* | *PF-KSWAPD* | *PF-MEMALLOC-NOFS* | *PF-MEMALLOC-NOIO*
| *PF-LESS-THROTTLE* | *PF-KTHREAD*
| *PF-RANDOMIZE* | *PF-SWAPWRITE* | *PF-NO-SETAFFINITY* | *PF-MCE-EARLY* | *PF-MUTEX-TESTER*
| *PF-FREEZER-SKIP* | *PF-SUSPEND-TASK*

record *fs-struct* = *users* ::*int*
umask :: *int*
in-exec :: *int*

record *files-struct* = *count* :: *int*

record *Task* = *real-cred* :: *Cred*
cred :: *Cred*
flags :: *Process-flags*
comm :: *string*
ptrace :: *int*
parent :: *process-id*
ptracer-cred :: *Cred option*

mm :: *mm-struct*
fs :: *fs-struct*
files :: *files-struct*
personality :: *nat*

record *rlimit* = *rlim-cur* :: *kernel-ulong-t*
rlim-max :: *kernel-ulong-t*

record *kern-ipc-perm* = *lock* :: *spinlock-t*
deleted :: *int*
id :: *int*
key :: *int*
k-uid :: *kuid*
k-gid :: *kgid*
cuid :: *kuid*
cgid :: *kgid*
mode :: *nat*
seq :: *nat*

record *msg-queue* = *q-perm* :: *kern-ipc-perm*

record *dentry* = *d-flags* :: *flags*
d-parent :: *dname*
d-sb :: *super-block*
d-inode :: *inum*
d-name :: *string*

record *ocfs2-security-xattr-info* = *enable* :: *int*
oname :: *string*
vvalue :: *string*
value-len :: *nat*

record *reiserfs-security-handle* = *rsh-name* ::*string*
rsh-value :: *string*
rsh-len :: *nat*

type-synonym *initxattrs* = *int*
type-synonym *time64-t* = *int*
type-synonym *long* = *int*

record *timespec64* = *tv-sec* :: *time64-t*
tv-nsec :: *long*

type-synonym *ts* = *timespec64*
typedecl *timezone*
type-synonym *tz* = *timezone*

record *iattr* = *ia-valid* :: *nat*
ia-mode :: *mode*
ia-uid :: *kuid*
ia-gid :: *kgid*
ia-size :: *loff-t*
ia-atime :: *timespec64*
ia-mtime :: *timespec64*
ia-ctime :: *timespec64*

typedecl *posix-acl*
typedecl *inode-operations*
typedecl *address-space*

record *nfs4-label* = *len* :: *nat*
label :: *string*

record *inode* = *i-mode* :: *mode*
i-opflags :: *flags*
i-uid :: *kuid-t*
i-gid :: *kgid-t*
i-flags :: *flags*
i-sb :: *super-block*
i-ino :: *nat*
i-acl :: *posix-acl*
i-default-acl :: *posix-acl*
i-op :: *inode-operations*
i-mapping :: *address-space*
ii-size :: *loff-t*

record *vfsmount* = *mnt-root* :: *dentry*
mnt-sb :: *super-block*
mnt-flags :: *int*

record *mount* = *mnt-mountpoint* :: *dentry*
mnt :: *vfsmount*

record *path* = *p-mnt* :: *vfsmount*
p-dentry :: *dentry*

record *binder-proc* = *tsk* :: *Task*

record *binder-thread* = *proc* :: *binder-proc*
typedecl *flat-binder-object*

record *binder-transaction* = *to-proc* :: *binder-proc*

datatype *file-operations* = *fop-read* | *fop-write* | *fop-mmap-capabilities*

record *Files* = *f-inode* :: *inode*

f-mode :: *mode*
f-path :: *path*
f-cred :: *Cred*
f-owner :: *fown-struct*
private-data :: *binder-proc*
f-op :: *file-operations*

record *fd* = *fdfile* :: *Files*
fd-flags :: *nat*

record *linux-binprm* = *called-set-creds* :: *int*
lfiles :: *Files*
lcred :: *Cred*
unsafe :: *int*
per-clear :: *nat*
typedecl *user-struct*
typedecl *ipc-namespace*
typedecl *ipc-params*

record *shmid-kernel* = *shm-perm* :: *kern-ipc-perm*
shm-file :: *Files*
shm-nattch :: *nat*
shm-segsz :: *nat*
shm-creator :: *Task*
shm-cprid :: *process-id*
shm-lprid :: *process-id*
mlock-user :: *user-struct*

record *sem-array* = *sem-perm* :: *kern-ipc-perm*
sem-nsems :: *int*
complex-count :: *int*

typedecl *delayed-call*

record *saved* = *saved-link* :: *path*
saved-done :: *delayed-call*
saved-name :: *string*
saved-seq :: *nat*

record *nameidata* = *nd-path* :: *path*
nd-root :: *path*
nd-last :: *string*
nd-inode :: *inode*
depth :: *nat*
nd-saved :: *saved*
link-inode :: *inode*
root-seq :: *nat*
nd-dfd :: *int*
stack :: *saved list*

nd-flags :: *nat*

typedecl *Port*
record *in-addr* = *s-addr* :: *nat*
typedecl *in6-addr*
typedecl *kernel-sa-family-t*

record *sockaddr-in* = *sin-port* :: *Port*
sin-addr :: *in-addr*
sin-family :: *kernel-sa-family-t*

type-synonym *ushort* = *nat*
record *sockaddr-in6* = *sin6-port* :: *Port*
sin6-addr :: *in6-addr*
sin6-family :: *ushort*
typedecl *sk*
datatype *Sk-Family* = *PF-INET* | *PF-INET6* | *AF-INET* | *AF-INET6* | *PF-UNSPEC*
| *PF-UNIX*

record *sock* = *sk-type* :: *nat*
sk-family :: *Sk-Family*
sk-socket :: *socketdesp*

record *sockaddr* = *sa-family* :: *ushort*
sa-data :: *string*

typedecl *socket-state*
typedecl *socket-wq*

record *proto-ops* = *proto-family* :: *int*

record *socket* = *skt-type* :: *int*
sk-flags :: *nat*
sk :: *sock option*
skt-state :: *socket-state*
wq :: *socket-wq*
skt-file :: *Files*
skt-ops :: *proto-ops*

record *socket-alloc* = *socket* :: *socket*
skvfs-inode :: *inode*

datatype *Msghdr-name* = *Sockaddr-in sockaddr-in* | *Sockaddr-in6 sockaddr-in6*

record *iov-iter* = *iov-type* :: *int*
iov-offset :: *nat*
iov-count :: *nat*

record *msghdr* = *msg-name* :: *Msghdr-name option* *msg-iter* :: *iov-iter*

typedecl *netlbl-lsm-secattr-catmap*

record *mls* = *lvl* :: *nat*
cat :: *netlbl-lsm-secattr-catmap*

record *attr* = *mls* :: *mls*
secid :: *nat*

record *netlbl-lsm-secattr* = *n-flags* :: *flags*
attr :: *attr*
type-synonym *u32* = *nat*
record *sk-buff* = *protocol* :: *int*
secmark :: *u32*
skb-iif :: *int*
typedecl *short*
record *sembuf* = *sem-num* :: *ushort*
sem-op :: *short*
sem-flg :: *short*

record *request-sock* = *secid* :: *u32*
peer-secid :: *u32*

record *ovl-cattr* = *mode* :: *mode*
link :: *string*
hardlink :: *dentry*

typedecl *fscache-cache*

record *fscache-object* = *fsobj-cache* :: *fscache-cache*

record *cachefiles-object* = *fscache* :: *fscache-object*
co-dentry :: *dentry*
backer :: *dentry*
i-size :: *loff-t*
co-type :: *nat*

record *cachefiles-xattr* = *cx-len* :: *nat*
cx-type :: *nat*

record *cachefiles-cache* = *cc-mnt* :: *vfsmount*
cache :: *fscache-cache*
graveyard :: *dentry*
cachefilesd :: *Files*
cache-cred :: *Cred*

typedecl *work-struct*

typedecl *xfrm-sec-ctx*
typedecl *xfrm-user-sec-ctx*
typedecl *xfrm-state*

record *bpf-map* = *work* :: *work-struct*

typedecl *rcu-head*

record *bpf-prog-aux* = *rcu* :: *rcu-head*

record *bpf-prog* = *bpf-len* :: *u32*
jited-len :: *u32*
aux :: *bpf-prog-aux*
typedecl *bpf-attr*

typedecl *xfrm-policy*

record *audit-context-ipc* = *audit-context-ipc-osid* :: *u32*

record *audit-context* = *dummy* :: *int*
in-syscall :: *int*
serial :: *int*
major :: *int*
ipc :: *audit-context-ipc*

type-synonym *kct* = *kernel-cap-t*
record *key* = *usage* ::*int*

record *security-mnt-opts* = *mnt-opts* :: *string list*
mnt-opts-flags :: *int list*
num-mnt-opts :: *int*
type-synonym *opts* = *security-mnt-opts*

record *nfs-parsed-mount-data* = *lsm-opts* :: *opts*

record *nfs-clone-mount* = *nfsc-sb* :: *super-block*

record *nfs-mount-info* = *parsed* :: *nfs-parsed-mount-data*
cloned :: *nfs-clone-mount*
typedecl *btrfs-fs-info*
typedecl *gfp-t*
typedecl *flowi*

type-synonym *key-ref-t* = *keyid*

datatype *enum-audit* = *Audit-equal* | *Audit-not-equal* | *Audit-bitmask* | *Audit-bittest*
| *Audit-lt*

| *Audit-gt* | *Audit-le* | *Audit-ge* | *Audit-bad*

record *audit-field* = *atype* :: *nat*
aop :: *enum-audit*
lsm-rule :: *string*
lsm-str :: *string*

record *audit-krule* = *field-count* :: *u32*
afields :: *audit-field list*

record *kernfs-iattrs* =
ia-iattr :: *iattr*
ia-secdata :: *string*
ia-secdata-len :: *nat*

record *ppid* = *level* :: *int*
tid :: *process-id*

record *proc-inode* = *vfs-inode* :: *inode*
proci-pid :: *ppid*

definition *EPERM* ≡ *1*
definition *ENOENT* ≡ *2*
definition *ESRCH* ≡ *3*
definition *EINTR* ≡ *4*
definition *EIO* ≡ *5*
definition *ENXIO* ≡ *6*
definition *E2BIG* ≡ *7*
definition *ENOEXEC* ≡ *8*
definition *EBADF* ≡ *9*
definition *ECHILD* ≡ *10*
definition *EAGAIN* ≡ *11*
definition *ENOMEM* ≡ *12*
definition *EACCES* ≡ *13*
definition *EFAULT* ≡ *14*
definition *ENOTBLK* ≡ *15*
definition *EBUSY* ≡ *16*
definition *EEXIST* ≡ *17*
definition *EXDEV* ≡ *18*
definition *ENODEV* ≡ *19*
definition *ENOTDIR* ≡ *20*
definition *EISDIR* ≡ *21*
definition *EINVAL* ≡ *22*
definition *ENFILE* ≡ *23*
definition *EMFILE* ≡ *24*
definition *ENOTTY* ≡ *25*
definition *ETXTBSY* ≡ *26*
definition *EFBIG* ≡ *27*

definition *ENOSPC* ≡ *28*
definition *ESPIPE* ≡ *29*
definition *EROFS* ≡ *30*
definition *EMLINK* ≡ *31*
definition *EPIPE* ≡ *32*
definition *EDOM* ≡ *33*
definition *ERANGE* ≡ *34*

1.7 audit$_h$

definition *AUDIT-GET* ≡ *1000*
definition *AUDIT-SET* ≡ *1001*
definition *AUDIT-LIST* ≡ *1002*
definition *AUDIT-ADD* ≡ *1003*
definition *AUDIT-DEL* ≡ *1004*
definition *AUDIT-USER* ≡ *1005*
definition *AUDIT-LOGIN* ≡ *1006*
definition *AUDIT-WATCH-INS* ≡ *1007*
definition *AUDIT-WATCH-REM* ≡ *1008*
definition *AUDIT-WATCH-LIST* ≡ *1009*
definition *AUDIT-SIGNAL-INFO* ≡ *1010*
definition *AUDIT-ADD-RULE* ≡ *1011*
definition *AUDIT-DEL-RULE* ≡ *1012*
definition *AUDIT-LIST-RULES* ≡ *1013*
definition *AUDIT-TRIM* ≡ *1014*
definition *AUDIT-MAKE-EQUIV* ≡ *1015*
definition *AUDIT-TTY-GET* ≡ *1016*
definition *AUDIT-TTY-SET* ≡ *1017*
definition *AUDIT-SET-FEATURE* ≡ *1018*
definition *AUDIT-GET-FEATURE* ≡ *1019*

consts *audit-sig-sid*:: *int*

definition *AUDIT-PID* ≡ *0*
definition *AUDIT-UID* ≡ *1*
definition *AUDIT-EUID* ≡ *2*
definition *AUDIT-SUID* ≡ *3*
definition *AUDIT-FSUID* ≡ *4*
definition *AUDIT-GID* ≡ *5*
definition *AUDIT-EGID* ≡ *6*
definition *AUDIT-SGID* ≡ *7*
definition *AUDIT-FSGID* ≡ *8*
definition *AUDIT-LOGINUID* ≡ *9*
definition *AUDIT-PERS* ≡*10*
definition *AUDIT-ARCH* ≡ *11*
definition *AUDIT-MSGTYPE* ≡ *12*
definition *AUDIT-SUBJ-USER* ≡ *13*
definition *AUDIT-SUBJ-ROLE* ≡ *14*
definition *AUDIT-SUBJ-TYPE* ≡ *15*

definition *AUDIT-SUBJ-SEN* ≡ *16*
definition *AUDIT-SUBJ-CLR* ≡ *17*
definition *AUDIT-PPID* ≡ *18*
definition *AUDIT-OBJ-USER* ≡ *19*
definition *AUDIT-OBJ-ROLE* ≡ *20*
definition *AUDIT-OBJ-TYPE* ≡ *21*
definition *AUDIT-OBJ-LEV-LOW* ≡ *22*
definition *AUDIT-OBJ-LEV-HIGH* ≡ *23*
definition *AUDIT-LOGINUID-SET* ≡ *24*
definition *AUDIT-SESSIONID* ≡ *25*
definition *AUDIT-FSTYPE* ≡ *26*

type-synonym *cap* = *int*

type-synonym *mm* = *mm-struct*

typedecl *seq-file*

type-synonym *qstr* = *string*

typedecl *dev-t*

definition *PTRACE-MODE-SET* = {*1*,*2*,*4*,*8*,*16*}

type-synonym *mask* = *int*

datatype *Void* = *String string* | *Int int*

definition *S-PRIVATE* = *512*

security define

definition *LSM-SETID-ID* = *1*
definition *LSM-SETID-RE* = *2*
definition *LSM-SETID-RES* = *4*
definition *LSM-SETID-FS* = *8*
definition *LSM-PRLIMIT-READ* = *1*
definition *LSM-PRLIMIT-WRITE* = *2*
definition *LSM-UNSAFE-SHARE* = *1*
definition *LSM-UNSAFE-PTRACE* = *2*
definition *LSM-UNSAFE-NO-NEW-PRIVS* = *4*
definition *ENOSYS*= *78*

definition *FS-OPEN-PERM* ≡ *0x00010000*
definition *FS-ACCESS-PERM* ≡ *0x00020000*

definition *PTRACE-MODE-READ* ≡ *0x01*
definition *PTRACE-MODE-ATTACH* ≡ *0x02*
definition *PTRACE-MODE-NOAUDIT* ≡ *4*

definition *PTRACE-MODE-FSCREDS* $\equiv$ *0x08*
definition *PTRACE-MODE-REALCREDS* $\equiv$ *0x10*

definition *MAY-EXEC* $\equiv$ *1*
definition *MAY-WRITE* $\equiv$ *2*
definition *MAY-READ* $\equiv$ *4*
definition *MAY-APPEND* $\equiv$ *8*
definition *MAY-ACCESS* $\equiv$ *0x00000010*
definition *MAY-OPEN* $\equiv$ *32*
definition *MAY-CHDIR* $\equiv$ *0x00000040*

definition *MAY-NOT-BLOCK* $\equiv$ *0x00000080*
definition*F-GETLK* $\equiv$ *7*
definition*F-SETLK* $\equiv$ *8*
definition*F-SETLKW* $\equiv$ *9*

definition*F-SETOWN* $\equiv$ *5*
definition*F-GETOWN* $\equiv$ *6*
definition*F-SETSIG* $\equiv$ *10*
definition*F-GETSIG* $\equiv$ *11*

definition*F-RDLCK* $\equiv$ *1*
definition*F-WRLCK* $\equiv$ *2*
definition*F-UNLCK* $\equiv$ *8*

definition*F-EXLCK* $\equiv$ *16*
definition*F-SHLCK* $\equiv$ *32*
definition *CAP-SYS-RAWIO* $\equiv$ *17*
definition *CAP-SYS-PTRACE* $\equiv$ *19*
definition *CAP-SYS-ADMIN* $\equiv$ *21*
definition *CAP-MAC-ADMIN* $\equiv$ *33*
definition *CAP-SETFCAP* $\equiv$ *31*
definition *CAP-MAC-OVERRIDE* $\equiv$ *32*
definition *SOCKFS-MAGIC* $\equiv$ *1397703499*
definition *EOPNOTSUPP* $\equiv$ *45*

definition *FMODE-READ* = *1*
definition *FMODE-WRITE* = *2*

definition *IS-PRIVATE* :: *inode* $\Rightarrow$ *int*
where *IS-PRIVATE i* $\equiv$ *i-flags i AND S-PRIVATE*

definition *RENAME-EXCHANGE* $\equiv$ *2*

definition *unlikely exp* $\equiv$ *if exp* = *0 then False else True*

datatype *xattr* = *XATTR-NAME-SMACK* | *XATTR-NAME-SMACKIPIN* | *XATTR-NAME-SMACKIPOU*
| *XATTR-NAME-SMACKEXEC*
| *XATTR-NAME-SMACKMMAP* | *XATTR-NAME-SMACKTRANSMUTE* |
XATTR-SMACK-SUFFIX | *XATTR-SMACK-IPIN*
| *XATTR-SMACK-IPOUT* | *XATTR-SMACK-EXEC* | *XATTR-SMACK-TRANSMUTE*
| *XATTR-SMACK-MMAP*
| *XATTR-SECURITY-PREFIX* | *XATTR-NAME-CAPS*

datatype *IOC-DIR* = *IOC-NONE* | *IOC-READ* | *IOC-WRITE* | *IOC-READWRITE*

datatype *fcntl-cmd* = *F-DUPFD* | *F-GETFD*| *F-SETFD* | *F-GETFL*

definition *ETH-P-IP* ≡ *2048*
definition *ETH-P-IPV6* ≡ *34525*

definition *S-NOSEC* ≡ *4096*
definition *IOP-XATTR* ≡ *8*
definition *SECURITY-CAP-AUDIT* ≡ *1*

definition *PAGE-SHIFT* ≡ *13*
definition *PAGE-SIZE* ≡ *4096*

definition *LOOKUP-FOLLOW* ≡ *0x0001*
definition *LOOKUP-DIRECTORY* ≡ *0x0002*
definition *LOOKUP-AUTOMOUNT* ≡ *0x0004*

definition *LOOKUP-PARENT* ≡ *0x0010*
definition *LOOKUP-REVAL* ≡ *0x0020*
definition *LOOKUP-RCU* ≡ *0x0040*
definition *LOOKUP-NO-REVAL* ≡ *0x0080*

definition *LOOKUP-OPEN* ≡ *0x0100*
definition *LOOKUP-CREATE* ≡ *0x0200*
definition *LOOKUP-EXCL* ≡ *0x0400*
definition *LOOKUP-RENAME-TARGET* ≡ *0x0800*

definition *LOOKUP-JUMPED* ≡ *0x1000*
definition *LOOKUP-ROOT* ≡ *0x2000*
definition *LOOKUP-EMPTY* ≡ *0x4000*
definition *LOOKUP-DOWN* ≡ *0x8000*

record *audit-names* = *hidden* :: *bool*
an-dev :: *dev-t*
osid :: *u32*

record *ovl-copy-up-ctx* = *copy-parent* :: *dentry*
copy-dentry :: *dentry*

definition *KREAD* ≡ *0*
definition *KWRITE* ≡ *1*

definition *SHM-RDONLY* ≡ *010000*
definition *SHM-RND* ≡ *020000*
definition *SHM-REMAP* ≡ *040000*
definition *SHM-EXEC* ≡ *0100000*

definition *SHM-LOCK* ≡ *11*
definition *SHM-UNLOCK* ≡ *12*

definition *SHM-STAT* ≡ *13*
definition *SHM-INFO* ≡ *14*
definition *SHM-STAT-ANY* ≡ *15*

definition *PROT-READ* ≡ *0x1*
definition *PROT-WRITE* ≡ *0x2*
definition *PROT-EXEC* ≡ *0x4*
definition *PROT-SEM* ≡ *0x8*
definition *PROT-NONE* ≡ *0x0*
definition *PROT-GROWSDOWN* ≡ *0x01000000*
definition *PROT-GROWSUP* ≡ *0x02000000*

definition *MAP-SHARED* ≡ *0x01*
definition *MAP-PRIVATE* ≡ *0x02*
definition *MAP-SHARED-VALIDATE* ≡ *0x03*
definition *MAP-TYPE* ≡ *0x0f*
definition *MAP-FIXED* ≡ *0x100*
definition *MAP-ANONYMOUS* ≡ *0x10*

definition *VM-NONE* ≡ *0x00000000*

definition *VM-READ* ≡ *0x00000001*
definition *VM-WRITE* ≡ *0x00000002*
definition *VM-EXEC* ≡ *0x00000004*
definition *VM-SHARED* ≡ *0x00000008*

definition *VM-MAYREAD* ≡ *0x00000010*
definition *VM-MAYWRITE* ≡ *0x00000020*
definition *VM-MAYEXEC* ≡ *0x00000040*
definition *VM-MAYSHARE* ≡ *0x00000080*

definition *VM-GROWSDOWN* ≡ *0x00000100*

definition *VM-UFFD-MISSING* ≡ *0x00000200*
definition *VM-PFNMAP* ≡ *0x00000400*
definition *VM-DENYWRITE* ≡ *0x00000800*
definition *VM-UFFD-WP* ≡ *0x00001000*

definition *VM-LOCKED* ≡ *0x00002000*
definition *VM-IO* ≡ *0x00004000*

definition *VM-SEQ-READ* ≡ *0x00008000*
definition *VM-RAND-READ* ≡ *0x00010000*

definition *VM-DONTCOPY* ≡ *0x00020000*
definition *VM-DONTEXPAND* ≡ *0x00040000*
definition *VM-LOCKONFAULT* ≡ *0x00080000*
definition *VM-ACCOUNT* ≡ *0x00100000*
definition *VM-NORESERVE* ≡ *0x00200000*
definition *VM-HUGETLB* ≡ *0x00400000*
definition *VM-SYNC* ≡ *0x00800000*
definition *VM-ARCH-1* ≡ *0x01000000*
definition *VM-WIPEONFORK* ≡ *0x02000000*
definition *VM-DONTDUMP* ≡ *0x04000000*

definition *UNAME26* ≡ *0x0020000*
definition *ADDR-NO-RANDOMIZE* ≡ *0x0040000*
definition *FDPIC-FUNCPTRS* ≡ *0x0080000*

definition *MMAP-PAGE-ZERO* ≡ *0x0100000*
definition *ADDR-COMPAT-LAYOUT* ≡ *0x0200000*
definition *READ-IMPLIES-EXEC* ≡ *0x0400000*
definition *ADDR-LIMIT-32BIT* ≡ *0x0800000*
definition *SHORT-INODE* ≡ *0x1000000*
definition *WHOLE-SECONDS* ≡ *0x2000000*
definition *STICKY-TIMEOUTS* ≡ *0x4000000*
definition *ADDR-LIMIT-3GB* ≡ *0x8000000*

definition *FDPUT-FPUT* ≡ *1*
definition *FDPUT-POS-UNLOCK* ≡ *2*

definition *SOL-SOCKET* ≡*0xffff*
definition *SO-DEBUG* ≡*0x0001*
definition *SO-REUSEADDR* ≡*0x0004*
definition *SO-KEEPALIVE* ≡*0x0008*
definition *SO-DONTROUTE* ≡*0x0010*
definition *SO-BROADCAST* ≡*0x0020*
definition *SO-LINGER* ≡*0x0080*

definition *SO-OOBINLINE* ≡*0x0100*
definition *SO-REUSEPORT* ≡*0x0200*
definition *SO-TYPE* ≡*0x1008*
definition *SO-ERROR* ≡*0x1007*
definition *SO-SNDBUF* ≡*0x1001*
definition *SO-RCVBUF* ≡*0x1002*
definition *SO-SNDBUFFORCE* ≡*0x100a*
definition *SO-RCVBUFFORCE* ≡*0x100b*
definition *SO-RCVLOWAT* ≡*0x1010*
definition *SO-SNDLOWAT* ≡*0x1011*
definition *SO-RCVTIMEO* ≡*0x1012*
definition *SO-SNDTIMEO* ≡*0x1013*
definition *SO-ACCEPTCONN* ≡*0x1014*
definition *SO-PROTOCOL* ≡*0x1028*
definition *SO-DOMAIN* ≡*0x1029*
definition *SO-NO-CHECK* ≡*11*
definition *SO-PRIORITY* ≡*12*
definition *SO-BSDCOMPAT* ≡*14*
definition *SO-PASSCRED* ≡*17*
definition *SO-PEERCRED* ≡*18*
definition *SO-BINDTODEVICE* ≡*25*
definition *SO-ATTACH-FILTER* ≡ *26*
definition *SO-DETACH-FILTER* ≡ *27*
definition *SO-GET-FILTER* ≡*SO-ATTACH-FILTER*
definition *SO-PEERNAME* ≡*28*
definition *SO-TIMESTAMP* ≡ *29*
definition *SCM-TIMESTAMP* ≡ *SO-TIMESTAMP*
definition *SO-PEERSEC* ≡*30*
definition *SO-PASSSEC* ≡*34*
definition *SO-TIMESTAMPNS* ≡ *35*
definition *SCM-TIMESTAMPNS* ≡ *SO-TIMESTAMPNS*
definition *SO-SECURITY-AUTHENTICATION* ≡*19*
definition *SO-SECURITY-ENCRYPTION-TRANSPORT* ≡*20*
definition *SO-SECURITY-ENCRYPTION-NETWORK* ≡*21*
definition *SO-MARK* ≡*36*
definition *SO-TIMESTAMPING* ≡*37*
definition *SCM-TIMESTAMPING* ≡*SO-TIMESTAMPING*
definition *SO-RXQ-OVFL* ≡ *40*
definition *SO-WIFI-STATUS* ≡*41*
definition *SCM-WIFI-STATUS* ≡*SO-WIFI-STATUS*
definition *SO-PEEK-OFF* ≡*42*
definition *SO-NOFCS* ≡*43*
definition *SO-LOCK-FILTER* ≡*44*
definition *SO-SELECT-ERR-QUEUE* ≡*45*
definition *SO-BUSY-POLL* ≡ *46*
definition *SO-MAX-PACING-RATE* ≡*47*
definition *SO-BPF-EXTENSIONS* ≡*48*
definition *SO-INCOMING-CPU* ≡ *49*
definition *SO-ATTACH-BPF* ≡ *50*

definition *SO-DETACH-BPF* ≡ *SO-DETACH-FILTER*
definition *SO-ATTACH-REUSEPORT-CBPF* ≡ *51*
definition *SO-ATTACH-REUSEPORT-EBPF* ≡ *52*
definition *SO-CNX-ADVICE* ≡ *53*
definition *SCM-TIMESTAMPING-OPT-STATS* ≡ *54*
definition *SO-MEMINFO* ≡ *55*
definition *SO-INCOMING-NAPI-ID* ≡ *56*
definition *SO-COOKIE* ≡ *57*
definition *SCM-TIMESTAMPING-PKTINFO* ≡ *58*
definition *SO-PEERGROUPS* ≡ *59*
definition *SO-ZEROCOPY* ≡ *60*
definition *SO-TXTIME* ≡ *61*
definition *SCM-TXTIME* ≡ *SO-TXTIME*

typedecl *kmem-cache*
record *proto* = *slab* :: *kmem-cache*

record *scm-cookie* = *scm-pid* :: *ppid*
scm-secid :: *u32*

record *sctp-ep-common* = *sctp-ep-sk* :: *sock*

record *sctp-endpoint* = *sctp-base* :: *sctp-ep-common*

record *sctp-chunk* = *sctp-skb* :: *sk-buff*

definition *SECMARK-MODE-SEL* ≡ *0x01*
definition *SECMARK-SECCTX-MAX* ≡ *256*

record *xt-secmark-target-info* = *xt-mode* :: *u8*
xt-secid :: *u32*
xt-secctx :: *string*

consts *xt-mode* :: *char*

record *tun-struct* = *numqueues* :: *nat*
tun-flags :: *nat*
tun-owner :: *kuid-t*
tun-group :: *kgid-t*

typedecl *ifreq*

record *key-type* = *kname* :: *string*

type-synonym *int32-t* = *int*
type-synonym *key-serial-t* = *int32-t*
type-synonym *key-perm-t* = *nat*

typedecl *Nlmsg-type*
record *nlmsghdr* = *nlmsg-len* :: *nat*
nlmsg-type :: *nat*

record *nf-conn* = *nf-secmark* :: *nat*

record *netlbl-audit* = *netlbl-audit-secid* :: *u32*
loginuid :: *kuid-t*
sessionid :: *nat*

record *kernfs-node* = *kn-iattr* :: *kernfs-iattrs*
kn-mode :: *mode*
kn-flags :: *nat*

record *gfs2-inode* = *i-inode* :: *inode*

record *svc-fh* = *fh-dentry* :: *dentry*

record *xdr-netobj* = *xdr-len* :: *nat*
xdr-data :: *string*
typedecl *nfs-fh*

definition *MNT-NOEXEC* ≡ *0x04*

definition *path-noexec* :: *path* ⇒*bool*
where *path-noexec p* ≡ ((*mnt-flags* (*p-mnt p*)) *AND MNT-NOEXEC*) ≠ *0*
∨ ((*int*(*s-iflags* (*mnt-sb* (*p-mnt p*)))) *AND SB-I-NOEXEC*) ≠ *0*
consts *CONFIG-MMU*::*bool*

definition *NOMMU-MAP-EXEC* ≡ *VM-MAYEXEC*

record *xattrs* = *xattr-name* :: *string*
xattr-value :: *string*
xattr-value-len :: *nat*

datatype *IPC-CMD* = *IPC-STAT* | *MSG-STAT* | *MSG-STAT-ANY* | *IPC-SET*
| *IPC-RMID* | *IPC-INFO* | *MSG-INFO*|
GETPID | *GETNCNT* | *GETZCNT* | *GETVAL* | *GETALL* | *SEM-STAT* | *SEM-STAT-ANY*

| *SETVAL* | *SETALL* | *SEM-INFO*
| *SHM-STAT* | *SHM-STAT-ANY* | *SHM-LOCK*| *SHM-UNLOCK*

typedecl *net*
typedecl *audit-buffer*

1.8 dache$_h$

definition *DCACHE-OP-HASH* ≡ *0x00000001*
definition *DCACHE-OP-COMPARE* ≡ *0x00000002*
definition *DCACHE-OP-REVALIDATE* ≡ *0x00000004*
definition *DCACHE-OP-DELETE* ≡ *0x00000008*
definition *DCACHE-OP-PRUNE* ≡ *0x00000010*
definition *DCACHE-DISCONNECTED* ≡ *0x00000020*

definition *DCACHE-REFERENCED* ≡ *0x00000040*

definition *DCACHE-CANT-MOUNT* ≡ *0x00000100*
definition *DCACHE-GENOCIDE* ≡ *0x00000200*
definition *DCACHE-SHRINK-LIST* ≡ *0x00000400*
definition *DCACHE-OP-WEAK-REVALIDATE* ≡ *0x00000800*
definition *DCACHE-NFSFS-RENAMED* ≡ *0x00001000*
definition *DCACHE-COOKIE* ≡ *0x00002000*
definition *DCACHE-FSNOTIFY-PARENT-WATCHED* ≡ *0x00004000*
definition *DCACHE-DENTRY-KILLED* ≡ *0x00008000*
definition *DCACHE-MOUNTED* ≡ *0x00010000*
definition *DCACHE-NEED-AUTOMOUNT* ≡ *0x00020000*
definition *DCACHE-MANAGE-TRANSIT* ≡ *0x00040000*

definition *DCACHE-LRU-LIST* ≡ *0x00080000*

definition *DCACHE-ENTRY-TYPE* ≡ *0x00700000*
definition *DCACHE-MISS-TYPE* ≡ *0x00000000*
definition *DCACHE-WHITEOUT-TYPE* ≡ *0x00100000*
definition *DCACHE-DIRECTORY-TYPE* ≡ *0x00200000*
definition *DCACHE-AUTODIR-TYPE* ≡ *0x00300000*
definition *DCACHE-REGULAR-TYPE* ≡ *0x00400000*
definition *DCACHE-SPECIAL-TYPE* ≡ *0x00500000*
definition *DCACHE-SYMLINK-TYPE* ≡ *0x00600000*

definition *DCACHE-MAY-FREE* ≡ *0x00800000*
definition *DCACHE-FALLTHRU* ≡ *0x01000000*
definition *DCACHE-ENCRYPTED-WITH-KEY* ≡ *0x02000000*
definition *DCACHE-OP-REAL* ≡ *0x04000000*

definition *DCACHE-PAR-LOOKUP* ≡ *0x10000000*
definition *DCACHE-DENTRY-CURSOR* ≡ *0x20000000*
definition *DCACHE-NORCU* ≡ *0x40000000*

definition *d-entry-type* :: *dentry* ⇒ *int*
where $\textit{d-entry-type } d \equiv \textit{d-flags } d \textit{ AND DCACHE-ENTRY-TYPE}$

definition *d-entry-type'* :: *dentry* ⇒ *int*
where $\textit{d-entry-type}' \textit{ dentry} \equiv (\textit{d-flags dentry AND DCACHE-ENTRY-TYPE})$

definition *d-is-autodir* :: *dentry* ⇒ *bool*
where $\textit{d-is-autodir dentry} \equiv \textit{if } (\textit{d-entry-type}' \textit{ dentry} = \textit{DCACHE-AUTODIR-TYPE}) \textit{ then True else False}$

definition *d-can-lookup* :: *dentry* ⇒ *bool*
where $\textit{d-can-lookup dentry} \equiv \textit{if } (\textit{d-entry-type}' \textit{ dentry} = \textit{DCACHE-DIRECTORY-TYPE}) \textit{ then True else False}$

definition *d-is-dir* :: *dentry* ⇒ *bool*
where $\textit{d-is-dir dentry} \equiv \textit{d-can-lookup dentry} \vee \textit{d-is-autodir dentry}$

definition *d-is-miss* :: *dentry* ⇒ *bool*
where $\textit{d-is-miss dentry} \equiv \textit{if } (\textit{d-entry-type}' \textit{ dentry} = \textit{DCACHE-MISS-TYPE}) \textit{ then True else False}$

definition *d-is-negative*:: *dentry* ⇒ *bool*
where $\textit{d-is-negative } d \equiv \textit{d-is-miss}(d)$

definition *d-is-positive*:: *dentry* ⇒ *bool*
where $\textit{d-is-positive } d \equiv \neg(\textit{d-is-negative } d)$

definition *d-is-whiteout* :: *dentry* ⇒ *bool*
where $\textit{d-is-whiteout dentry} \equiv \textit{if } (\textit{d-entry-type}' \textit{ dentry} = \textit{DCACHE-WHITEOUT-TYPE}) \textit{ then True else False}$

definition *d-is-symlink* :: *dentry* ⇒ *bool*
where $\textit{d-is-symlink dentry} \equiv \textit{if } (\textit{d-entry-type}' \textit{ dentry} = \textit{DCACHE-SYMLINK-TYPE}) \textit{ then True else False}$

definition *d-is-reg* :: *dentry* ⇒ *bool*
where $\textit{d-is-reg dentry} \equiv \textit{if } (\textit{d-entry-type}' \textit{ dentry} = \textit{DCACHE-REGULAR-TYPE}) \textit{ then True else False}$

definition *d-is-special* :: *dentry* ⇒ *bool*
where $\textit{d-is-special dentry} \equiv \textit{if } (\textit{d-entry-type}' \textit{ dentry} = \textit{DCACHE-SPECIAL-TYPE}) \textit{ then True else False}$

definition *d-is-file*:: *dentry* ⇒ *bool*
where $\textit{d-is-file dentry} \equiv \textit{d-is-reg dentry} \vee \textit{d-is-special dentry}$

definition *d-is-fallthru*:: *dentry* ⇒ *bool*
where $\textit{d-is-fallthru dentry} \equiv \textit{if } (\textit{d-flags dentry AND DCACHE-FALLTHRU}) \neq$

0 then True else False

definition *d-inodeid* :: *dentry* ⇒ *inum*
 where *d-inodeid dentry* ≡ (*d-inode dentry*)

1.9 fcntl$_h$

definition *F-LINUX-SPECIFIC-BASE* ≡ *1024*
definition *F-SETLEASE* ≡(*F-LINUX-SPECIFIC-BASE* + *0*)
definition *F-GETLEASE* ≡(*F-LINUX-SPECIFIC-BASE* + *1*)

definition *F-CANCELLK* ≡(*F-LINUX-SPECIFIC-BASE* + *5*)

definition *F-DUPFD-CLOEXEC* ≡(*F-LINUX-SPECIFIC-BASE* + *6*)

definition *F-NOTIFY* ≡(*F-LINUX-SPECIFIC-BASE*+*2*)

definition *F-SETPIPE-SZ* ≡(*F-LINUX-SPECIFIC-BASE* + *7*)
definition *F-GETPIPE-SZ* ≡(*F-LINUX-SPECIFIC-BASE* + *8*)

definition *F-ADD-SEALS* ≡(*F-LINUX-SPECIFIC-BASE* + *9*)
definition *F-GET-SEALS* ≡(*F-LINUX-SPECIFIC-BASE* + *10*)

definition *F-SEAL-SEAL* ≡ *0x0001*
definition *F-SEAL-SHRINK* ≡ *0x0002*
definition *F-SEAL-GROW* ≡ *0x0004*
definition *F-SEAL-WRITE* ≡ *0x0008*

definition *F-GET-RW-HINT* ≡ (*F-LINUX-SPECIFIC-BASE* + *11*)
definition *F-SET-RW-HINT* ≡ (*F-LINUX-SPECIFIC-BASE* + *12*)
definition *F-GET-FILE-RW-HINT* ≡ (*F-LINUX-SPECIFIC-BASE* + *13*)
definition *F-SET-FILE-RW-HINT* ≡ (*F-LINUX-SPECIFIC-BASE* + *14*)

definition *RWF-WRITE-LIFE-NOT-SET* ≡ *0*
definition *RWH-WRITE-LIFE-NONE* ≡ *1*
definition *RWH-WRITE-LIFE-SHORT* ≡ *2*
definition *RWH-WRITE-LIFE-MEDIUM* ≡ *3*
definition *RWH-WRITE-LIFE-LONG* ≡ *4*
definition *RWH-WRITE-LIFE-EXTREME* ≡ *5*

definition *DN-ACCESS* ≡ *0x00000001*
definition *DN-MODIFY* ≡ *0x00000002*
definition *DN-CREATE* ≡ *0x00000004*
definition *DN-DELETE* ≡ *0x00000008*
definition *DN-RENAME* ≡ *0x00000010*
definition *DN-ATTRIB* ≡ *0x00000020*
definition *DN-MULTISHOT* ≡ *0x80000000*

definition *AT-FDCWD* ≡ *−100*
definition *AT-SYMLINK-NOFOLLOW* ≡ *0x100*
definition *AT-REMOVEDIR* ≡ *0x200*
definition *AT-SYMLINK-FOLLOW* ≡ *0x400*
definition *AT-NO-AUTOMOUNT* ≡ *0x800*
definition *AT-EMPTY-PATH* ≡ *0x1000*

definition *AT-STATX-SYNC-TYPE* ≡ *0x6000*
definition *AT-STATX-SYNC-AS-STAT* ≡ *0x0000*
definition *AT-STATX-FORCE-SYNC* ≡ *0x2000*
definition *AT-STATX-DONT-SYNC* ≡ *0x4000*
definition *Q-SYNC* ≡ *8388609*
end

2 The Core of the Subject-Object Access Control Policy For Smack

In this theory, we introduce subject-object access control policy. A subject is an active entity, usually a process (running program), that causes information to flow among objects or changes the system state. On Smack a subject is a task, which is in turn the basic unit of execution. An object is a passive entity that contains or receives data, such as a File,Inode, IPC, Sock. A process may be an object, such as when you use kill on a process. All subjects and objects in a system have labels. The label determines which information you can access.

theory
  SOAC
  imports
    Element
begin

2.1 Model of a AC configuration

datatype *decision* = *allow* | *deny*

datatype *access* = *READ* | *WRITE* | *EXECUTE* | *APPEND* | *T* | *LOCK* | *Control* | *OWN*

datatype *Request* = *MAY-WRITE′* | *MAY-READ′* | *MAY-EXECUTE′* | *MAY-APPEND′*
| *MAY-T′* | *MAY-LOCK′*

type-synonym (*′subj*,*′obj*) *policy-table* = *′subj*⇒(*′subj* × *′obj*) ⇒ *access set*

Label:Data that identifies the Mandatory Access Control characteristics of a subject or an object. The format of an access rule is: subject$_{l}abelobject_{l}abelaccess.Eachrulemustspec$
$Unclassforunclassified, Cforclassified, Sforsecret, andTSfortopsecret.Then, withahandfulofru$
C Unclass rx S C rx S Unclass rx TS S rx TS C rx TS Unclass rx the traditional hierarchy of access is defined. Because of the Smack defaults, Unclass will only be able to access data with that same label, whereas because of the rules above, TS can access S, C and Unclass data. Note that there is no transitivity in Smack rules,just because S can access C and TS can access S, that does not mean that TS can access C.

datatype *Label* = *Normal string* | *Floor* | *Hat* | *Star* | *Huh* | *Web* | *UNDEFINED*

type-synonym *subject-label* = *Label*
type-synonym *object-label* = *Label*

type-synonym *Subj* = *process-id*
datatype *Obj* = *Sb super-block* | *Process process-id* | *File Files* | *IPC kern-ipc-perm*
| *Msg msg-msg*
| *ObjInode inode* | *ObjSock sock* | *ObjKey key*

definition *access-rl* :: *Request* => *access*
where *access-rl r* ≡ (*case r of MAY-WRITE′* ⇒ *WRITE* |
MAY-READ′ ⇒ *READ* |
MAY-EXECUTE′ ⇒ *EXECUTE* |
MAY-APPEND′ ⇒ *APPEND* |
MAY-T′ ⇒ *T* |
MAY-LOCK′ ⇒ *LOCK*
)

locale *SOModel* =
fixes *subj-label* :: *′s* ⇒*Subj* ⇒ *subject-label*
fixes *obj-label* :: *′s* ⇒*Obj* ⇒*object-label*
fixes *access-rules* :: *Label* ⇒ *Label* ⇒ *access set*
fixes *Subj* :: *′s* ⇒ *Subj set*
fixes *Obj* :: *′s* ⇒ *Obj set*
fixes *request* :: *′s*⇒*Subj* ⇒ *Obj* ⇒ *Request*⇒ *decision*

begin

abbreviation *subjects-have-auth* :: *Subj* ⇒ *access* ⇒ *bool*
where *subjects-have-auth subj a* ≡ ∀ *s obj*. *subj* ∈ *Subj s*⟶ *a* ∈ *access-rules* (*subj-label s subj*) (*obj-label s obj*)

end

end

theory *Value-Abbreviation*

imports *Main*

keywords *value-abbreviation* :: *thy-decl*

begin

Computing values and saving as abbreviations.

Useful in program verification to handle some configuration constant (e.g. n = 4) which may change. This mechanism can be used to give names (abbreviations) to other related constants (e.g. $2^{\,n}, 2^n - 1, [1..n], rev[1..n]) which may appear repeatedly.$

ML ‹
```
structure Value-Abbreviation = struct
fun value-and-abbreviation mode name expr int ctxt = let
    val decl = (name, NONE, Mixfix.NoSyn)
    val expr = Syntax.read-term ctxt expr
    val eval-expr =Value-Command.value ctxt expr
    val lhs = Free (Binding.name-of name, fastype-of expr)
    val eq = Logic.mk-equals (lhs, eval-expr)
    val ctxt = Specification.abbreviation mode (SOME decl) [] eq int ctxt
    val pretty-eq = Syntax.pretty-term ctxt eq
  in Pretty.writeln pretty-eq; ctxt end

val - =
  Outer-Syntax.local-theory' @{command-keyword value-abbreviation}
    setup abbreviation for evaluated value
    (Parse.syntax-mode -- Parse.binding -- Parse.term
      >> (fn ((mode, name), expr) => value-and-abbreviation mode name expr));

end
```
›

Testing it out. Unfortunately locale/experiment/notepad all won't work here because the code equation setup is all global.

definition
value-abbreviation-test-config-constant-1 = (*24* :: *nat*)

definition
value-abbreviation-test-config-constant-2 = (*5* :: *nat*)

value-abbreviation (*input*)
value-abbreviation-test-important-magic-number

((2 :: int) ^ value-abbreviation-test-config-constant-1)
− (2 ^ value-abbreviation-test-config-constant-2)

value-abbreviation (*input*)
value-abbreviation-test-range-of-options
rev [int value-abbreviation-test-config-constant-2
.. int value-abbreviation-test-config-constant-1]

end

theory *Match-Abbreviation*

imports *Main*

keywords *match-abbreviation* :: *thy-decl*
and *reassoc-thm* :: *thy-decl*

begin

Splicing components of terms and saving as abbreviations. See the example at the bottom for explanation/documentation.

ML ‹

```
structure Match-Abbreviation = struct

fun app-cons-dummy cons x y
  = Const (cons, dummyT) $ x $ y

fun lazy-lam x t = if Term.exists-subterm (fn t' => t' aconv x) t
    then lambda x t else t

fun abs-dig-f ctxt lazy f (Abs (nm, T, t))
  = let
    val (nms, ctxt) = Variable.variant-fixes [nm] ctxt
    val x = Free (hd nms, T)
    val t = betapply (Abs (nm, T, t), x)
    val t' = f ctxt t
  in if lazy then lazy-lam x t' else lambda x t' end
  | abs-dig-f - - - t = raise TERM (abs-dig-f: not abs, [t])

fun find-term1 ctxt get (f $ x)
  = (get ctxt (f $ x) handle Option => (find-term1 ctxt get f
      handle Option => find-term1 ctxt get x))
  | find-term1 ctxt get (a as Abs -)
  = abs-dig-f ctxt true (fn ctxt => find-term1 ctxt get) a
  | find-term1 ctxt get t = get ctxt t

fun not-found pat t = raise TERM (pattern not found, [pat, t])

fun find-term ctxt get pat t = find-term1 ctxt get t
```

```
  handle Option => not-found pat t

fun lambda-frees-vars ctxt ord-t t = let
    fun is-free t = is-Free t andalso not (Variable.is-fixed ctxt (Term.term-name t))
    fun is-it t = is-free t orelse is-Var t
    val get = fold-aterms (fn t => if is-it t then insert (=) t else I)
    val all-vars = get ord-t []
    val vars = get t []
    val ord-vars = filter (member (=) vars) all-vars
  in fold lambda ord-vars t end

fun parse-pat-fixes ctxt fixes pats = let
    val (-, ctxt') = Variable.add-fixes
          (map (fn (b, -, -) => Binding.name-of b) fixes) ctxt
    val read-pats = Syntax.read-terms ctxt' pats
  in Variable.export-terms ctxt' ctxt read-pats end

fun add-reassoc name rhs fixes thms-info ctxt = let
    val thms = Attrib.eval-thms ctxt thms-info
    val rhs-pat = singleton (parse-pat-fixes ctxt fixes) rhs
      |> Thm.cterm-of ctxt
    val rew = Simplifier.rewrite (clear-simpset ctxt addsimps thms) rhs-pat
      |> Thm.symmetric
    val (-, ctxt) = Local-Theory.note ((name, []), [rew]) ctxt
    val pretty-decl = Pretty.block [Pretty.str (Binding.name-of name ^ :\n),
        Thm.pretty-thm ctxt rew]
  in Pretty.writeln pretty-decl; ctxt end

fun dig-f ctxt repeat adj (f $ x) = (adj ctxt (f $ x)
    handle Option => (dig-f ctxt repeat adj f
          $ (if repeat then (dig-f ctxt repeat adj x
              handle Option => x) else x)
        handle Option => f $ dig-f ctxt repeat adj x))
  | dig-f ctxt repeat adj (a as Abs -)
    = abs-dig-f ctxt false (fn ctxt => dig-f ctxt repeat adj) a
  | dig-f ctxt - adj t = adj ctxt t

fun do-rewrite ctxt repeat rew-pair t = let
    val thy = Proof-Context.theory-of ctxt
    fun adj - t = case Pattern.match-rew thy t rew-pair
      of NONE => raise Option | SOME (t', -) => t'
  in dig-f ctxt repeat adj t
    handle Option => not-found (fst rew-pair) t end

fun select-dig ctxt [] f t = f ctxt t
  | select-dig ctxt (p :: ps) f t = let
    val thy = Proof-Context.theory-of ctxt
    fun do-rec ctxt t = if Pattern.matches thy (p, t)
      then select-dig ctxt ps f t else raise Option
```

```
  in dig-f ctxt false do-rec t handle Option => not-found p t end

fun ext-dig-lazy ctxt f (a as Abs -)
  = abs-dig-f ctxt true (fn ctxt => ext-dig-lazy ctxt f) a
  | ext-dig-lazy ctxt f t = f ctxt t

fun report-adjust ctxt nm t = let
    val pretty-decl = Pretty.block [Pretty.str (nm ^ , have:\n),
        Syntax.pretty-term ctxt t]
  in Pretty.writeln pretty-decl; t end

fun do-adjust ctxt (((select, []), [p]), fixes) t = let
    val p = singleton (parse-pat-fixes ctxt fixes) p
    val thy = Proof-Context.theory-of ctxt
    fun get - t = if Pattern.matches thy (p, t) then t else raise Option
    val t = find-term ctxt get p t
  in report-adjust ctxt Selected t end
  | do-adjust ctxt (((retype-consts, []), consts), []) t = let
    fun get-constname (Const (s, -)) = s
      | get-constname (Abs (-, -, t)) = get-constname t
      | get-constname (f $ -) = get-constname f
      | get-constname - = raise Option
    fun get-constname2 t = get-constname t
      handle Option => raise TERM (do-adjust: no constant, [t])
    val cnames = map (get-constname2 o Syntax.read-term ctxt) consts
      |> Symtab.make-set
    fun adj (Const (cn, T)) = if Symtab.defined cnames cn
        then Const (cn, dummyT) else Const (cn, T)
      | adj t = t
    val t = Syntax.check-term ctxt (Term.map-aterms adj t)
  in report-adjust ctxt Adjusted types t end
  | do-adjust ctxt (((r, in-selects), [from, to]), fixes) t = if
        r = rewrite1 orelse r = rewrite then let
    val repeat = r <> rewrite1
    val sel-pats = map (fn (p, fixes) => singleton (parse-pat-fixes ctxt fixes) p)
        in-selects
    val rewrite-pair = case parse-pat-fixes ctxt fixes [from, to]
      of [f, t] => (f, t) | - => error (do-adjust: unexpected length)
    val t = ext-dig-lazy ctxt (fn ctxt => select-dig ctxt sel-pats
        (fn ctxt => do-rewrite ctxt repeat rewrite-pair)) t
  in report-adjust ctxt (if repeat then Rewrote else Rewrote (repeated)) t end
  else error (do-adjust: unexpected:  ^ r)
  | do-adjust - args - = error (do-adjust: unexpected:  ^ @{make-string} args)

fun unvarify-types-same ty = ty
  |> Term-Subst.map-atypsT-same
    (fn TVar ((a, i), S) => TFree (a ^ -var- ^ string-of-int i, S)
      | - => raise Same.SAME)
```

```
fun unvarify-types tm = tm
  |> Same.commit (Term-Subst.map-types-same unvarify-types-same)

fun match-abbreviation mode name init adjusts int ctxt = let
    val init-term = init ctxt
    val init-lambda = lambda-frees-vars ctxt init-term init-term
      |> unvarify-types
      |> Syntax.check-term ctxt
    val decl = (name, NONE, Mixfix.NoSyn)
    val result = fold (do-adjust ctxt) adjusts init-lambda
    val lhs = Free (Binding.name-of name, fastype-of result)
    val eq = Logic.mk-equals (lhs, result)
    val ctxt = Specification.abbreviation mode (SOME decl) [] eq int ctxt
    val pretty-eq = Syntax.pretty-term ctxt eq
  in Pretty.writeln pretty-eq; ctxt end

fun from-thm f thm-info ctxt = let
    val thm = singleton (Attrib.eval-thms ctxt) thm-info
  in f thm end

fun from-term term-str ctxt = Syntax.parse-term ctxt term-str

val init-term-parse = Parse.$$$ in |--
    ((Parse.reserved concl |-- Parse.thm >> from-thm Thm.concl-of)
        || (Parse.reserved thm-prop |-- Parse.thm >> from-thm Thm.prop-of)
        || (Parse.term >> from-term)
    )

val term-to-term = (Parse.term -- (Parse.reserved to |-- Parse.term))
    >> (fn (a, b) => [a, b])

val p-for-fixes = Scan.optional
    (Parse.$$$ ( |-- Parse.for-fixes --| Parse.$$$ )) []

val adjust-parser = Parse.and-list1
     ((Parse.reserved select -- Scan.succeed [] -- (Parse.term >> single) --
p-for-fixes)
        || (Parse.reserved retype-consts -- Scan.succeed []
            -- Scan.repeat Parse.term -- Scan.succeed [])
        || ((Parse.reserved rewrite1 || Parse.reserved rewrite)
            -- Scan.repeat (Parse.$$$ in |-- Parse.term -- p-for-fixes)
            -- term-to-term -- p-for-fixes)
    )

(* install match-abbreviation. see below for examples/docs *)
val - =
  Outer-Syntax.local-theory' @{command-keyword match-abbreviation}
    setup abbreviation for subterm of theorem
    (Parse.syntax-mode -- Parse.binding
```

```
    -- init-term-parse -- adjust-parser
  >> (fn (((mode, name), init), adjusts)
      => match-abbreviation mode name init adjusts));

val - =
  Outer-Syntax.local-theory @{command-keyword reassoc-thm}
    store a reassociate-theorem
    (Parse.binding -- Parse.term -- p-for-fixes -- Scan.repeat Parse.thm
      >> (fn (((name, rhs), fixes), thms)
          => add-reassoc name rhs fixes thms));
end
⟩
```

The match/abbreviate command. There are examples of all elements below, and an example involving monadic syntax in the theory Match-Abbreviation-Test.

Each invocation is match abbreviation, a syntax mode (e.g. (input)), an abbreviation name, a term specifier, and a list of adjustment specifiers.

A term specifier can be term syntax or the conclusion or proposition of some theorem. Examples below.

Each adjustment is a select, a rewrite, or a constant retype.

The select adjustment picks out the part of the term matching the pattern (examples below). It picks the first match point, ordered in term order with compound terms before their subterms and functions before their arguments.

The rewrite adjustment uses a pattern pair, and rewrites instances of the first pattern into the second. The match points are found in the same order as select. The "in" specifiers (examples below) limit the rewriting to within some matching subterm, specified with pattern in the same way as select. The rewrite1 variant only rewrites once, at the first matching site.

The rewrite mechanism can be used to replace terms with terms of different types. The retype adjustment can then be used to repair the term by resetting the types of all instances of the named constants. This is used below with list constructors, to assemble a new list with a different element type.

experiment begin

Fetching part of the statement of a theorem.

match-abbreviation (*input*) *fixp-thm-bit*
  in *thm-prop fixp-induct-tailrec*
  select $X \equiv Y$ (**for** X Y)

Ditto conclusion.

match-abbreviation (*input*) *rev-simps-bit*
  in *concl rev.simps*(*2*)
  select X (**for** X)

Selecting some conjuncts and reorienting an equality.

match-abbreviation (*input*) *conjunct-test*
in $(P \wedge Q \wedge P \wedge P \wedge P \wedge ((1 :: nat) = 2) \wedge Q \wedge Q, [Suc\ 0,\ 0])$
select $Q \wedge Z$ (**for** Z)
and *rewrite* $x = y$ *to* $y = x$ (**for** $x\ y$)
and *rewrite* **in** $x = y\ \&\ Z$ (**for** $x\ y\ Z$)
$A \wedge B$ *to* A (**for** $A\ B$)

The relevant reassociate theorem, that rearranges a conjunction like the above to group the elements selected.

reassoc-thm *conjunct-test-reassoc*
conjunct-test $P\ Q \wedge Z$ (**for** $P\ Q\ Z$)
conj-assoc

Selecting some elements of a list, and then replacing tuples with equalities, and adjusting the type of the list constructors so the new term is type correct.

match-abbreviation (*input*) *list-test*
in $[(Suc\ 1,\ Suc\ 2),\ (4,\ 5),\ (6,\ 7),\ (8,\ 9),\ (10,\ 11),\ (x,\ y),\ (6,\ 7),$
$(18,\ 19),\ a,\ a,\ a,\ a,\ a,\ a,\ a]$
select $(4,\ V)\ \#\ xs$ (**for** $V\ xs$)
and *rewrite* $(x,\ y)$ *to* $(y,\ x)$ (**for** $x\ y$)
and *rewrite1* **in** $(9,\ V)\ \#\ xs$ (**for** $V\ xs$) **in** $(7,\ V)\ \#\ xs$ (**for** $V\ xs$)
$x\ \#\ xs$ *to* $[x]$ (**for** $x\ xs$)
and *rewrite* $(x,\ y)$ *to* $x = y$ (**for** $x\ y$)
and *retype-consts Cons Nil*

end

end

theory *Subgoal-Methods*
imports *Main*
begin
ML ‹

```
signature SUBGOAL-METHODS =
sig
  val fold-subgoals: Proof.context -> bool -> thm -> thm
  val unfold-subgoals-tac: Proof.context -> tactic
  val distinct-subgoals: Proof.context -> thm -> thm
end;

structure Subgoal-Methods: SUBGOAL-METHODS =
struct

fun max-common-prefix eq (ls :: lss) =
      let
```

```
      val ls' = tag-list 0 ls;
      fun all-prefix (i,a) =
        forall (fn ls' => if length ls' > i then eq (a, nth ls' i) else false) lss
      val ls'' = take-prefix all-prefix ls'
    in map snd ls'' end
  | max-common-prefix - [] = [];

fun push-outer-params ctxt th =
  let
    val ctxt' = ctxt
      |> Simplifier.empty-simpset
      |> Simplifier.add-simp Drule.norm-hhf-eq;
  in
    Conv.fconv-rule
      (Raw-Simplifier.rewrite-cterm (true, false, false) (K (K NONE)) ctxt') th
  end;

fun fix-schematics ctxt raw-st =
  let
    val ((schematic-types, [st']), ctxt1) = Variable.importT [raw-st] ctxt;
    val ((-, inst), ctxt2) =
      Variable.import-inst true [Thm.prop-of st'] ctxt1;

    val schematic-terms = map (apsnd (Thm.cterm-of ctxt2)) inst;
    val schematics = (schematic-types, schematic-terms);

  in (Thm.instantiate schematics st', ctxt2) end

val strip-params = Term.strip-all-vars;
val strip-prems = Logic.strip-imp-prems o Term.strip-all-body;
val strip-concl = Logic.strip-imp-concl o Term.strip-all-body;



fun fold-subgoals ctxt prefix raw-st =
  if Thm.nprems-of raw-st < 2 then raw-st
  else
    let
      val (st, inner-ctxt) = fix-schematics ctxt raw-st;

      val subgoals = Thm.prems-of st;
      val paramss = map strip-params subgoals;
      val common-params = max-common-prefix (eq-snd (op =)) paramss;

      fun strip-shift subgoal =
        let
          val params = strip-params subgoal;
          val diff = length common-params − length params;
          val prems = strip-prems subgoal;
```

```
    in map (Term.incr-boundvars diff) prems end;

  val premss = map (strip-shift) subgoals;

  val common-prems = max-common-prefix (op aconv) premss;

  val common-params = if prefix then common-params else [];
  val common-prems = if prefix then common-prems else [];

  fun mk-concl subgoal =
    let
      val params = Term.strip-all-vars subgoal;
      val local-params = drop (length common-params) params;
      val prems = strip-prems subgoal;
      val local-prems = drop (length common-prems) prems;
      val concl = strip-concl subgoal;
    in Logic.list-all (local-params, Logic.list-implies (local-prems, concl)) end;

  val goal =
    Logic.list-all (common-params,
    (Logic.list-implies (common-prems,Logic.mk-conjunction-list (map mk-concl
subgoals))));

  val chyp = Thm.cterm-of inner-ctxt goal;

  val (common-params',inner-ctxt') =
    Variable.add-fixes (map fst common-params) inner-ctxt
    |>> map2 (fn (-, T) => fn x => Thm.cterm-of inner-ctxt (Free (x, T)))
common-params;

  fun try-dest rule =
     try (fn () => (@{thm conjunctionD1} OF [rule], @{thm conjunctionD2}
OF [rule])) ();

  fun solve-headgoal rule =
    let
      val rule' = rule
        |> Drule.forall-intr-list common-params'
        |> push-outer-params inner-ctxt';
    in
      (fn st => Thm.implies-elim st rule')
    end;

  fun solve-subgoals rule' st =
    (case try-dest rule' of
      SOME (this, rest) => solve-subgoals rest (solve-headgoal this st)
    | NONE => solve-headgoal rule' st);

  val rule = Drule.forall-elim-list common-params' (Thm.assume chyp);
```

```
    in
      st
      |> push-outer-params inner-ctxt
      |> solve-subgoals rule
      |> Thm.implies-intr chyp
      |> singleton (Variable.export inner-ctxt' ctxt)
    end;

fun distinct-subgoals ctxt raw-st =
  let
    val (st, inner-ctxt) = fix-schematics ctxt raw-st;
    val subgoals = Drule.cprems-of st;
    val atomize = Conv.fconv-rule (Object-Logic.atomize-prems inner-ctxt);

    val rules =
      map (atomize o Raw-Simplifier.norm-hhf inner-ctxt o Thm.assume) subgoals
      |> sort (int-ord o apply2 Thm.nprems-of);

    val st' = st
      |> ALLGOALS (fn i =>
        Object-Logic.atomize-prems-tac inner-ctxt i THEN solve-tac inner-ctxt rules
i)
      |> Seq.hd;

    val subgoals' = subgoals
      |> inter (op aconvc) (Thm.chyps-of st')
      |> distinct (op aconvc);
  in
    Drule.implies-intr-list subgoals' st'
    |> singleton (Variable.export inner-ctxt ctxt)
  end;

(* Variant of filter-prems-tac that recovers premise order *)
fun filter-prems-tac' ctxt pred =
  let
    fun Then NONE tac = SOME tac
      | Then (SOME tac) tac' = SOME (tac THEN' tac');
    fun thins H (tac, n, i) =
      (if pred H then (tac, n + 1, i)
       else (Then tac (rotate-tac n THEN' eresolve-tac ctxt [thin-rl]), 0, i + n));
  in
    SUBGOAL (fn (goal, i) =>
      let val Hs = Logic.strip-assums-hyp goal in
        (case fold thins Hs (NONE, 0, 0) of
          (NONE, -, -) => no-tac
        | (SOME tac, -, n) => tac i THEN rotate-tac (~ n) i)
      end)
  end;
```

```
fun trim-prems-tac ctxt rules =
let
  fun matches (prem,rule) =
  let
    val ((-,prem'),ctxt') = Variable.focus NONE prem ctxt;
    val rule-prop = Thm.prop-of rule;
  in Unify.matches-list (Context.Proof ctxt') [rule-prop] [prem'] end;

in filter-prems-tac' ctxt (not o member matches rules) end;

val adhoc-conjunction-tac = REPEAT-ALL-NEW
  (SUBGOAL (fn (goal, i) =>
    if can Logic.dest-conjunction (Logic.strip-imp-concl goal)
    then resolve0-tac [Conjunction.conjunctionI] i
    else no-tac));

fun unfold-subgoals-tac ctxt =
  TRY (adhoc-conjunction-tac 1)
  THEN (PRIMITIVE (Raw-Simplifier.norm-hhf ctxt));

val - =
  Theory.setup
   (Method.setup @{binding fold-subgoals}
      (Scan.lift (Args.mode prefix) >> (fn prefix => fn ctxt =>
         SIMPLE-METHOD (PRIMITIVE (fold-subgoals ctxt prefix))))
      lift all subgoals over common premises/params #>
    Method.setup @{binding unfold-subgoals}
      (Scan.succeed (fn ctxt => SIMPLE-METHOD (unfold-subgoals-tac ctxt)))
      recover subgoals after folding #>
    Method.setup @{binding distinct-subgoals}
    (Scan.succeed (fn ctxt => SIMPLE-METHOD (PRIMITIVE (distinct-subgoals
ctxt))))
     trim all subgoals to be (logically) distinct #>
    Method.setup @{binding trim}
      (Attrib.thms >> (fn thms => fn ctxt =>
         SIMPLE-METHOD (HEADGOAL (trim-prems-tac ctxt thms))))
     trim all premises that match the given rules);

end;
```
›

end

theory *Rule-By-Method*
imports
  Main
  HOL−Eisbach.Eisbach-Tools
begin

ML ‹

```
signature RULE-BY-METHOD =
sig
  val rule-by-tac: Proof.context -> {vars : bool, prop: bool} ->
    (Proof.context -> tactic) -> (Proof.context -> tactic) list -> Position.T
-> thm
end;


fun atomize ctxt = Conv.fconv-rule (Object-Logic.atomize ctxt);

fun fix-schematics ctxt raw-st =
  let
    val ((schematic-types, [st']), ctxt1) = Variable.importT [raw-st] ctxt;
    fun certify-inst ctxt inst = map (apsnd (Thm.cterm-of ctxt)) (#2 inst)
    val (schematic-terms, ctxt2) =
      Variable.import-inst true [Thm.prop-of st'] ctxt1
      |>> certify-inst ctxt1;
    val schematics = (schematic-types, schematic-terms);
  in (Thm.instantiate schematics st', ctxt2) end

fun curry-asm ctxt st = if Thm.nprems-of st = 0 then Seq.empty else
let

  val prems = Thm.cprem-of st 1 |> Thm.term-of |> Logic.strip-imp-prems;

  val (thesis :: xs,ctxt') = Variable.variant-fixes (thesis :: replicate (length prems)
P) ctxt;

  val rl =
    xs
    |> map (fn x => Thm.cterm-of ctxt' (Free (x, propT)))
    |> Conjunction.mk-conjunction-balanced
    |> (fn xs => Thm.apply (Thm.apply @{cterm Pure.imp} xs) (Thm.cterm-of
ctxt' (Free (thesis,propT))))
    |> Thm.assume
    |> Conjunction.curry-balanced (length prems)
    |> Drule.implies-intr-hyps

  val rl' = singleton (Variable.export ctxt' ctxt) rl;

  in Thm.bicompose (SOME ctxt) {flatten = false, match = false, incremented =
false}
            (false, rl', 1) 1 st end;

val drop-trivial-imp =
let
  val asm =
```

```
    Thm.assume (Drule.protect @{cprop (PROP A ==> PROP A) ==> PROP A})
    |> Goal.conclude;

in
  Thm.implies-elim  asm (Thm.trivial @{cprop PROP A})
  |> Drule.implies-intr-hyps
  |> Thm.generalize ([],[A]) 1
  |> Drule.zero-var-indexes
 end

val drop-trivial-imp' =
let
  val asm =
    Thm.assume (Drule.protect @{cprop (PROP P ==> A) ==> A})
    |> Goal.conclude;

  val asm' = Thm.assume @{cprop PROP P == Trueprop A}

in
  Thm.implies-elim asm (asm' COMP Drule.equal-elim-rule1)
  |> Thm.implies-elim (asm' COMP Drule.equal-elim-rule2)
  |> Drule.implies-intr-hyps
  |> Thm.permute-prems 0 ~1
  |> Thm.generalize ([],[A,P]) 1
  |> Drule.zero-var-indexes
 end

fun atomize-equiv-tac ctxt i =
  Object-Logic.full-atomize-tac ctxt i
  THEN PRIMITIVE (fn st'  =>
  let val (-,[A,-]) = Drule.strip-comb (Thm.cprem-of st' i) in
  if Object-Logic.is-judgment ctxt (Thm.term-of A) then st'
 else error (Failed to fully atomize result:\n ^ (Syntax.string-of-term ctxt (Thm.term-of
A))) end)


structure Data = Proof-Data
(
  type T = thm list * bool;
  fun init - = ([],false);
);

val empty-rule-prems = Data.map (K ([],true));

fun add-rule-prem thm = Data.map (apfst (Thm.add-thm thm));

fun with-rule-prems enabled parse =
  Scan.state :|-- (fn context =>
  let
```

```
    val context' = Context.proof-of context |> Data.map (K ([Drule.free-dummy-thm],enabled))
                   |> Context.Proof
  in Scan.lift (Scan.pass context' parse) end)


fun get-rule-prems ctxt =
  let
    val (thms,b) = Data.get ctxt
  in if (not b) then [] else thms end


fun zip-subgoal assume tac (ctxt,st : thm) = if Thm.nprems-of st = 0 then Se-
q.single (ctxt,st) else
let
  fun bind-prems st' =
  let
    val prems = Drule.cprems-of st';
    val (asms, ctxt') = Assumption.add-assumes prems ctxt;
    val ctxt'' = fold add-rule-prem asms ctxt';
    val st'' = Goal.conclude (Drule.implies-elim-list st' (map Thm.assume prems));
  in (ctxt'',st'') end

  fun defer-prems st' =
  let
    val nprems = Thm.nprems-of st';
    val st'' = Thm.permute-prems 0 nprems (Goal.conclude st');
  in (ctxt,st'') end;


in
  tac ctxt (Goal.protect 1 st)
  |> Seq.map (if assume then bind-prems else defer-prems)  end


fun zip-subgoals assume tacs pos ctxt st =
let
  val nprems = Thm.nprems-of st;
  val - = nprems < length tacs andalso error (More tactics than rule assumptions
^ Position.here pos);
  val tacs' = map (zip-subgoal assume) (tacs @ (replicate (nprems − length tacs)
(K all-tac)));
  val ctxt' = empty-rule-prems ctxt;
in Seq.EVERY tacs' (ctxt',st) end;

fun rule-by-tac' ctxt {vars,prop} tac asm-tacs pos raw-st =
  let
    val (st,ctxt1) = if vars then (raw-st,ctxt) else fix-schematics ctxt raw-st;

    val ([x],ctxt2) = Proof-Context.add-fixes [(Binding.name Auto-Bind.thesisN,NONE,
```

```
NoSyn)] ctxt1;

  val thesis = if prop then Free (x,propT) else Object-Logic.fixed-judgment ctxt2
x;

  val cthesis = Thm.cterm-of ctxt thesis;

  val revcut-rl′ = Thm.instantiate′ [] ([NONE,SOME cthesis]) @{thm revcut-rl};

  fun is-thesis t = Logic.strip-assums-concl t aconv thesis;

  fun err thm str = error (str ^ Position.here pos ^ \n ^
    (Pretty.string-of (Goal-Display.pretty-goal ctxt thm)));

  fun pop-thesis st =
  let
    val prems = Thm.prems-of st |> tag-list 0;
    val (i,-) = (case filter (is-thesis o snd) prems of
      [] => err st Lost thesis
      | [x] => x
      | - => err st More than one result obtained);
   in st |> Thm.permute-prems 0 i  end

  val asm-st =
  (revcut-rl′ OF [st])
  |> (fn st => Goal.protect (Thm.nprems-of st − 1) st)


  val (ctxt3,concl-st) = case Seq.pull (zip-subgoals (not vars) asm-tacs pos ctxt2
asm-st) of
      SOME (x,-) => x
  | NONE => error (Failed to apply tactics to rule assumptions. ^ (Position.here
pos));

  val concl-st-prepped =
    concl-st
    |> Goal.conclude
     |> (fn st => Goal.protect (Thm.nprems-of st) st |> Thm.permute-prems 0
~1 |> Goal.protect 1)

  val concl-st-result = concl-st-prepped
    |> (tac ctxt3
        THEN (PRIMITIVE pop-thesis)
        THEN curry-asm ctxt
          THEN PRIMITIVE (Goal.conclude #> Thm.permute-prems 0 1 #>
Goal.conclude))

  val result = (case Seq.pull concl-st-result of
    SOME (result,-) => singleton (Proof-Context.export ctxt3 ctxt) result
```

```
    | NONE => err concl-st-prepped Failed to apply tactic to rule conclusion:)

   val drop-rule = if prop then drop-trivial-imp else drop-trivial-imp'

   val result' = ((Goal.protect (Thm.nprems-of result -1) result) RS drop-rule)
   |> (if prop then all-tac else
      (atomize-equiv-tac ctxt (Thm.nprems-of result)
      THEN resolve-tac ctxt @{thms Pure.reflexive} (Thm.nprems-of result)))
   |> Seq.hd
   |> Raw-Simplifier.norm-hhf ctxt

  in Drule.zero-var-indexes result' end;

fun rule-by-tac is-closed ctxt args tac asm-tacs pos raw-st =
 let val f = rule-by-tac' ctxt args tac asm-tacs pos
  in
   if is-closed orelse Context-Position.is-really-visible ctxt then SOME (f raw-st)
   else try f raw-st
 end

fun pos-closure (scan : 'a context-parser) :
  (('a * (Position.T * bool)) context-parser) = (fn (context,toks) =>
  let
     val (((context',x),tr-toks),toks') = Scan.trace (Scan.pass context (Scan.state
-- scan)) toks;
    val pos = Token.range-of tr-toks;
    val is-closed = exists (fn t => is-some (Token.get-value t)) tr-toks
  in ((x,(Position.range-position pos, is-closed)),(context',toks')) end)

val parse-flags = Args.mode schematic -- Args.mode raw-prop >> (fn (b,b') =>
{vars = b, prop = b'})

fun tac m ctxt =
  Method.NO-CONTEXT-TACTIC ctxt
    (Method.evaluate-runtime m ctxt []);

(* Declare as a mixed attribute to avoid any partial evaluation *)

fun handle-dummy f (context, thm) =
  case (f context thm) of SOME thm' => (NONE, SOME thm')
  | NONE => (SOME context, SOME Drule.free-dummy-thm)

val (rule-prems-by-method : attribute context-parser) = Scan.lift parse-flags :--
(fn flags =>
  pos-closure (Scan.repeat1
    (with-rule-prems (not (#vars flags)) Method.text-closure ||
      Scan.lift (Args.$$$ - >> (K Method.succeed-text))))) >>
        (fn (flags,(ms,(pos, is-closed))) => handle-dummy (fn context =>
          rule-by-tac is-closed (Context.proof-of context) flags (K all-tac) (map tac
```

ms) *pos*))

val (*rule-concl-by-method* : *attribute context-parser*) = *Scan.lift parse-flags* :−−
(*fn flags* =>
  pos-closure (*with-rule-prems* (*not* (#*vars flags*)) *Method.text-closure*)) >>
    (*fn* (*flags*,(*m*,(*pos*, *is-closed*))) => *handle-dummy* (*fn context* =>
      rule-by-tac is-closed (*Context.proof-of context*) *flags* (*tac m*) [] *pos*))

val - = *Theory.setup*
  (*Global-Theory.add-thms-dynamic* (@{*binding rule-prems*},
    (*fn context* => *get-rule-prems* (*Context.proof-of context*))) #>
   Attrib.setup @{*binding* #} *rule-prems-by-method*
    transform rule premises with method #>
   Attrib.setup @{*binding* @} *rule-concl-by-method*
    transform rule conclusion with method #>
   Attrib.setup @{*binding atomized*}
    (*Scan.succeed* (*Thm.rule-attribute* []
      (*fn context* => *fn thm* =>
        Conv.fconv-rule (*Object-Logic.atomize* (*Context.proof-of context*)) *thm*
          |> *Drule.zero-var-indexes*)))
    atomize rule)
›

experiment begin

ML ‹
  val [*att*] = @{*attributes* [@‹*erule thin-rl*, *cut-tac TrueI*, *fail*›]}
  val k = *Attrib.attribute* @{*context*} *att*
  val - = *case* (*try k* (*Context.Proof* @{*context*}, *Drule.dummy-thm*)) *of*
    SOME - => *error Should fail*
    | - => ()
  ›

lemmas *baz* = [[@‹*erule thin-rl*, *rule revcut-rl*[*of* $P \longrightarrow P \wedge P$], *simp*›]] **for** P

lemmas *bazz*[*THEN impE*] = *TrueI*[@‹*erule thin-rl*, *rule revcut-rl*[*of* $P \longrightarrow P \wedge P$], *simp*›] **for** P

lemma $Q \longrightarrow Q \wedge Q$ **by** (*rule baz*)

method *silly-rule* **for** P :: *bool* **uses** *rule* =
  (*rule* [[@‹*erule thin-rl*, *cut-tac rule*, *drule asm-rl*[*of P*]›]])

lemma assumes A **shows** A **by** (*silly-rule A rule*: ‹A›)

lemma assumes A[*simp*]: A **shows** A
  apply (*match* **conclusion in** P **for** $P \Rightarrow$
      ‹*rule* [[@‹*erule thin-rl*, *rule revcut-rl*[*of P*], *simp*›]]›)
  done

end

end

theory *Local-Method*
imports *Main*
keywords *supply-local-method* :: *prf-script* % *proof*
begin

See documentation in `Local_Method_Tests.thy`.

ML ‹
```
  structure MethodData = Proof-Data(
    type T = Method.method Symtab.table
    val init = K Symtab.empty);
›
```

method-setup *local-method* = ‹
```
  Scan.lift Parse.liberal-name >>
  (fn name => fn - => fn facts => fn (ctxt, st) =>
    case (ctxt |> MethodData.get |> Symtab.lookup) name of
        SOME method => method facts (ctxt, st)
      | NONE => Seq.succeed (Seq.Error (K (Couldn't find method text named ^
quote name))))
›
```

ML ‹
```
local

val parse-name-text-ranges =
 Scan.repeat1 (Parse.liberal-name --| Parse.!!! @{keyword =} -- Method.parse)

fun supply-method-cmd name-text-ranges ctxt =
  let
    fun add-method ((name, (text, range)), ctxt) =
      let
        val - = Method.report (text, range)
        val method = Method.evaluate text ctxt
      in
        MethodData.map (Symtab.update (name, method)) ctxt
      end
  in
    List.foldr add-method ctxt name-text-ranges
  end

val - =
  Outer-Syntax.command @{command-keyword‹supply-local-method›}
    Add a local method alias to the current proof context
```

(*parse-name-text-ranges* >> (*Toplevel.proof o Proof.map-context o supply-method-cmd*))

in end
›

end

theory *Eisbach-Methods*
imports
subgoal-focus/Subgoal-Methods
HOL−Eisbach.Eisbach-Tools
Rule-By-Method
Local-Method
begin

3 Debugging methods

method *print-concl* = (*match* **conclusion in** *P* **for** *P* ⇒ ‹*print-term P*›)

method-setup *print-raw-goal* = ‹*Scan.succeed (fn ctxt => fn facts =>*
(fn (ctxt, st) => (Output.writeln (Thm.string-of-thm ctxt st);
Seq.make-results (Seq.single (ctxt, st)))))›

ML ‹*fun method-evaluate text ctxt facts =*
Method.NO-CONTEXT-TACTIC ctxt
(Method.evaluate-runtime text ctxt facts)›

method-setup *print-headgoal* =
‹*Scan.succeed (fn ctxt =>*
fn - => fn (ctxt′, thm) =>
((SUBGOAL (fn (t,-) =>
(Output.writeln
(Pretty.string-of (Syntax.pretty-term ctxt t)); all-tac)) 1 thm);
(Seq.make-results (Seq.single (ctxt′, thm)))))›

4 Simple Combinators

method-setup *defer-tac* = ‹*Scan.succeed (fn - => SIMPLE-METHOD (defer-tac 1))*›
method-setup *prefer-last* = ‹*Scan.succeed (fn - => SIMPLE-METHOD (PRIMITIVE (Thm.permute-prems 0 ~1)))*›

method-setup *all* =
‹*Method.text-closure >> (fn m => fn ctxt => fn facts =>*

```
  let
    fun tac i st' =
      Goal.restrict i 1 st'
      |> method-evaluate m ctxt facts
      |> Seq.map (Goal.unrestrict i)

  in SIMPLE-METHOD (ALLGOALS tac) facts end)
⟩
```

method-setup *determ* =

```
⟨Method.text-closure >> (fn m => fn ctxt => fn facts =>
  let
    fun tac st' = method-evaluate m ctxt facts st'

  in SIMPLE-METHOD (DETERM tac) facts end)
⟩ ⟨Run the given method, but only yield the first result⟩
```

ML ⟨

```
fun require-determ (method : Method.method) facts st =
  case method facts st |> Seq.filter-results |> Seq.pull of
    NONE => Seq.empty
  | SOME (r1, rs) =>
      (case Seq.pull rs of
         NONE => Seq.single r1 |> Seq.make-results
       | - => Method.fail facts st);

fun require-determ-method text ctxt =
  require-determ (Method.evaluate-runtime text ctxt);
⟩
```

method-setup *require-determ* =

```
  ⟨Method.text-closure >> require-determ-method⟩
  ⟨Run the given method, but fail if it returns more than one result⟩
```

method-setup *changed* =

```
⟨Method.text-closure >> (fn m => fn ctxt => fn facts =>
  let
    fun tac st' = method-evaluate m ctxt facts st'

  in SIMPLE-METHOD (CHANGED tac) facts end)
⟩
```

method-setup *timeit* =

```
⟨Method.text-closure >> (fn m => fn ctxt => fn facts =>
  let
    fun timed-tac st seq = Seq.make (fn () => Option.map (apsnd (timed-tac st))
      (timeit (fn () => (Seq.pull seq))));
```

fun tac st′ =
timed-tac st′ (method-evaluate m ctxt facts st′);

in SIMPLE-METHOD tac [] end)
⟩

method-setup *timeout* =
⟨*Scan.lift Parse.int −− Method.text-closure >> (fn (i,m) => fn ctxt => fn facts =>*
let
fun str-of-goal th = Pretty.string-of (Goal-Display.pretty-goal ctxt th);

fun limit st f x = Timeout.apply (Time.fromSeconds i) f x
handle Timeout.TIMEOUT - => error (Method timed out:\n ^ (str-of-goal st));

fun timed-tac st seq = Seq.make (limit st (fn () => Option.map (apsnd (timed-tac st))
(Seq.pull seq)));

fun tac st′ =
timed-tac st′ (method-evaluate m ctxt facts st′);

in SIMPLE-METHOD tac [] end)
⟩

method *repeat-new* **methods** *m* = (*m* ; (*repeat-new* ⟨*m*⟩)*?*)

The following *fails* and *succeeds* methods protect the goal from the effect of a method, instead simply determining whether or not it can be applied to the current goal. The *fails* method inverts success, only succeeding if the given method would fail.

method-setup *fails* =
⟨*Method.text-closure >> (fn m => fn ctxt => fn facts =>*
let
fun fail-tac st′ =
(case Seq.pull (method-evaluate m ctxt facts st′) of
SOME - => Seq.empty
| NONE => Seq.single st′)

in SIMPLE-METHOD fail-tac facts end)
⟩

method-setup *succeeds* =
⟨*Method.text-closure >> (fn m => fn ctxt => fn facts =>*
let
fun can-tac st′ =
(case Seq.pull (method-evaluate m ctxt facts st′) of

SOME (*st''*,-) => *Seq.single st'*
| *NONE* => *Seq.empty*)

in SIMPLE-METHOD can-tac facts end)
⟩

This method wraps up the "focus" mechanic of match without actually doing any matching. We need to consider whether or not there are any assumptions in the goal, as premise matching fails if there are none.

If the *fails* method is removed here, then backtracking will produce a set of invalid results, where only the conclusion is focused despite the presence of subgoal premises.

method *focus-concl* **methods** *m* =
((*fails* ⟨*erule thin-rl*⟩, *match* **conclusion in** - ⇒ ⟨*m*⟩)
| *match* **premises** (*local*) **in** *H*:- (*multi*) ⇒ ⟨*m*⟩)

repeat applies a method a specific number of times, like a bounded version of the '+' combinator.

usage: apply (repeat n *text*)

- Applies the method *text* to the current proof state n times. - Fails if *text* can't be applied n times.

ML ⟨
fun repeat-tac count tactic =
if count = *0*
then all-tac
else tactic THEN (*repeat-tac* (*count* − *1*) *tactic*)
⟩

method-setup *repeat* = ⟨
Scan.lift Parse.nat −− *Method.text-closure* >> (*fn* (*count*, *text*) => *fn ctxt* => *fn facts* =>
let val tactic = *method-evaluate text ctxt facts*
in SIMPLE-METHOD (*repeat-tac count tactic*) *facts end*)
⟩

notepad begin
fix *A B C*
assume *assms*: *A B C*

repeat: simple repeated application.

have *A* ∧ *B* ∧ *C* ∧ *True*

repeat: fails if method can't be applied the specified number of times.

apply (*fails* ⟨*repeat 4* ⟨*rule conjI*, *rule assms*⟩⟩)
apply (*repeat 3* ⟨*rule conjI*, *rule assms*⟩)
by (*rule TrueI*)

repeat: application with subgoals.

have $A \wedge A$ $B \wedge B$ $C \wedge C$
apply −

We have three subgoals. This *repeat* call consumes two of them.

apply (*repeat 2* ‹*rule conjI*, (*rule assms*)+›)

One subgoal remaining...

apply (*rule conjI*, (*rule assms*)+)
done

end

Literally a copy of the parser for *subgoal-tac* composed with an analogue of **prefer**.

Useful if you find yourself introducing many new facts via 'subgoal$_tac$', $butprefertoprovethemimmedia$

setup ‹
Method.setup ***binding*** ‹*prop-tac*›
(*Args.goal-spec* −− *Scan.lift* (*Scan.repeat1 Args.embedded-inner-syntax* −− *Parse.for-fixes*) >>
(*fn* (*quant*, (*props*, *fixes*)) => *fn ctxt* =>
(*SIMPLE-METHOD''* *quant*
(*EVERY'* (*map* (*fn prop* => *Rule-Insts.subgoal-tac ctxt prop fixes*) *props*)
THEN'
(*K* (*prefer-tac 2*))))))
insert prop (*dynamic instantiation*), *introducing prop subgoal first*
›

notepad begin {
fix *xs*
assume *assms*: *list-all even* (*xs* :: *nat list*)

from *assms* **have** *even* (*sum-list xs*)
apply (*induct xs*)
apply *simp*

Inserts the desired proposition as the current subgoal.

apply (*prop-tac list-all even xs*)
subgoal by *simp*

The prop *list-all even xs* is now available as an assumption. Let's add another one.

apply (*prop-tac even* (*sum-list xs*))
subgoal by *simp*

Now that we've proven our introduced props, use them!

apply *clarsimp*

```
  done
}
end
```

5 Advanced combinators

5.1 Protecting goal elements (assumptions or conclusion) from methods

```
context
begin

private definition protect-concl x ≡ ¬ x
private definition protect-false ≡ False

private lemma protect-start: (protect-concl P ⟹ protect-false) ⟹ P
  by (simp add: protect-concl-def protect-false-def) (rule ccontr)

private lemma protect-end: protect-concl P ⟹ P ⟹ protect-false
  by (simp add: protect-concl-def protect-false-def)

method only-asm methods m =
  (match premises in H[thin]:- (multi,cut) ⇒
    ‹rule protect-start,
     match premises in H'[thin]:protect-concl - ⇒
       ‹insert H,m;rule protect-end[OF H']››)

method only-concl methods m = (focus-concl ‹m›)

end

notepad begin
 fix D C
  assume DC:D ⟹ C
  have D ∧ D ⟹ C ∧ C
  apply (only-asm ‹simp›) — stash conclusion before applying method
  apply (only-concl ‹simp add: DC›) — hide premises from method
  by (rule DC)

end
```

5.2 Safe subgoal folding (avoids expanding meta-conjuncts)

Isabelle's goal mechanism wants to aggressively expand meta-conjunctions if they are the top-level connective. This means that *fold-subgoals* will immediately be unfolded if there are no common assumptions to lift over.

To avoid this we simply wrap conjunction inside of conjunction' to hide it from the usual facilities.

context begin

definition
conjunction' :: *prop* ⇒ *prop* ⇒ *prop* (**infixr** &^& *2*) **where**
conjunction' A B ≡ (*PROP A* &&& *PROP B*)

In general the context antiquotation does not work in method definitions.
Here it is fine because Conv.top$_sweep_convisjustover-specifiedtoneedaProof.contextwhenanything$

method *safe-meta-conjuncts* =
raw-tactic
‹*REPEAT-DETERM*
(*CHANGED-PROP*
(*PRIMITIVE*
(*Conv.gconv-rule* ((*Conv.top-sweep-conv* (*K* (*Conv.rewr-conv* @{*thm conjunction'-def*[*symmetric*]})) @{*context*})) *1*)))›

method *safe-fold-subgoals* = (*fold-subgoals* (*prefix*), *safe-meta-conjuncts*)

lemma *atomize-conj'* [*atomize*]: (*A* &^& *B*) == *Trueprop* (*A* & *B*)
by (*simp add*: *conjunction'-def*, *rule atomize-conj*)

lemma *context-conjunction'I*:
PROP P ⟹ (*PROP P* ⟹ *PROP Q*) ⟹ *PROP P* &^& *PROP Q*
apply (*simp add*: *conjunction'-def*)
apply (*rule conjunctionI*)
apply *assumption*
apply (*erule meta-mp*)
apply *assumption*
done

lemma *conjunction'I*:
PROP P ⟹ *PROP Q* ⟹ *PROP P* &^& *PROP Q*
by (*rule context-conjunction'I*; *simp*)

lemma *conjunction'E*:
assumes *PQ*: *PROP P* &^& *PROP Q*
assumes *PQR*: *PROP P* ⟹ *PROP Q* ⟹ *PROP R*
shows
PROP R
apply (*rule PQR*)
apply (*rule PQ*[*simplified conjunction'-def*, *THEN conjunctionD1*])
by (*rule PQ*[*simplified conjunction'-def*, *THEN conjunctionD2*])

end

notepad begin
fix *D C E*

assume *DC*: *D* ∧ *C*

have $D\ C \wedge C$
apply $-$
apply (*safe-fold-subgoals*, *simp*, *atomize* (*full*))
apply (*rule DC*)
done

end

6 Utility methods

6.1 Finding a goal based on successful application of a method

context begin

method-setup *find-goal* =
‹*Method.text-closure* >> (*fn m* => *fn ctxt* => *fn facts* =>
let
fun prefer-first i = *SELECT-GOAL*
(*fn st′* =>
(*case Seq.pull* (*method-evaluate m ctxt facts st′*) *of*
SOME (*st″*,-) => *Seq.single st″*
| *NONE* => *Seq.empty*)) *i THEN prefer-tac i*

in SIMPLE-METHOD (*FIRSTGOAL prefer-first*) *facts end*)›

end

notepad begin

fix *A B*
assume *A*: *A* **and** *B*: *B*

have *A A B*
apply (*find-goal* ‹*match conclusion in B* ⇒ ‹−››)
apply (*rule B*)
by (*rule A*)+

have $A \wedge A\ A \wedge A\ B$
apply (*find-goal* ‹*fails* ‹*simp*››) — find the first goal which cannot be simplified
apply (*rule B*)
by (*simp add*: *A*)+

have $B\ A\ A \wedge A$
apply (*find-goal* ‹*succeeds* ‹*simp*››) — find the first goal which can be simplified (without doing so)
apply (*rule conjI*)
by (*rule A B*)+

end

6.2 Remove redundant subgoals

Tries to solve subgoals by assuming the others and then using the given method. Backtracks over all possible re-orderings of the subgoals.

context begin

definition *protect* (*PROP P*) ≡ *P*

lemma *protectE*: *PROP protect P* ⟹ (*PROP P* ⟹ *PROP R*) ⟹ *PROP R* **by** (*simp add*: *protect-def*)

private lemmas *protect-thin* = *thin-rl*[**where** *V*=*PROP protect P* **for** *P*]

private lemma *context-conjunction'I-protected*:
  assumes *P*: *PROP P*
  assumes *PQ*: *PROP protect* (*PROP P*) ⟹ *PROP Q*
  shows
  PROP P &^& *PROP Q*
   apply (*simp add*: *conjunction'-def*)
   apply (*rule P*)
  apply (*rule PQ*)
  apply (*simp add*: *protect-def*)
  by (*rule P*)

private lemma *conjunction'-sym*: *PROP P* &^& *PROP Q* ⟹ *PROP Q* &^& *PROP P*
  apply (*simp add*: *conjunction'-def*)
  apply (*frule conjunctionD1*)
  apply (*drule conjunctionD2*)
  apply (*rule conjunctionI*)
  by *assumption+*

private lemmas *context-conjuncts'I* =
  context-conjunction'I-protected
  context-conjunction'I-protected[*THEN conjunction'-sym*]

method *distinct-subgoals-strong* **methods** *m* =
  (*safe-fold-subgoals*,
   (*intro context-conjuncts'I*;
    (((*elim protectE conjunction'E*)?, *solves* ⟨*m*⟩)
    | (*elim protect-thin*)?)))?

end

method *forward-solve* **methods** *fwd m* =
  (*fwd*, *prefer-last*, *fold-subgoals*, *safe-meta-conjuncts*, *rule conjunction'I*,
   defer-tac, ((*intro conjunction'I*)?; *solves* ⟨*m*⟩))[*1*]

method *frule-solve* **methods** m **uses** *rule* = (*forward-solve* ‹*frule rule*› ‹m›)
method *drule-solve* **methods** m **uses** *rule* = (*forward-solve* ‹*drule rule*› ‹m›)

notepad begin
{
fix $A\ B\ C\ D\ E$
assume $ABCD$: $A \Longrightarrow B \Longrightarrow C \Longrightarrow D$
assume ACD: $A \Longrightarrow C \Longrightarrow D$
assume DE: $D \Longrightarrow E$
assume $B\ C$

have $A \Longrightarrow D$
apply (*frule-solve* ‹*simp add*: ‹B› ‹C›› *rule*: $ABCD$)
apply (*drule-solve* ‹*simp add*: ‹B› ‹C›› *rule*: ACD)
apply (*match* **premises in** $A \Rightarrow$ ‹*fail*› | - $\Rightarrow$ ‹−›)
apply *assumption*
done
}
end

notepad begin
{
fix $A\ B\ C$
assume A: A
have $A\ B \Longrightarrow A$
apply −
apply (*distinct-subgoals-strong* ‹*assumption*›)
by (*rule* A)

have $B \Longrightarrow A\ A$
by (*distinct-subgoals-strong* ‹*assumption*›, *rule* A) — backtracking required here
}

{
fix $A\ B\ C$

assume B: B
assume BC: $B \Longrightarrow C\ B \Longrightarrow A$
have $A\ B \longrightarrow (A \wedge C)\ B$
apply (*distinct-subgoals-strong* ‹*simp*›, *rule* B) — backtracking required here
by (*simp add*: BC)

}
end

7 Attribute methods (for use with rule$_b y_m ethodattributes$)

method *prove-prop-raw* **for** P :: *prop* **methods** m =
(*erule thin-rl*, *rule revcut-rl*[*of PROP P*],
solves ‹*match conclusion in* - ⇒ ‹*m*››)

method *prove-prop* **for** P :: *prop* = (*prove-prop-raw PROP P* ‹*auto*›)

experiment begin

lemma assumes A[*simp*]:A **shows** A **by** (*rule* [[@‹*prove-prop A*›]])

end

8 Shortcuts for prove$_p rop.Notethesearelessefficientthanusingtherawsy$ $proveneverytime.$

method *ruleP* **for** P :: *prop* = (*catch* ‹*rule* [[@‹*prove-prop PROP P*›]]› ‹*fail*›)
method *insertP* **for** P :: *prop* = (*catch* ‹*insert* [[@‹*prove-prop PROP P*›]]› ‹*fail*›)[*1*]

experiment begin

lemma assumes A[*simp*]:A **shows** A **by** (*ruleP False* | *ruleP A*)
lemma assumes A:A **shows** A **by** (*ruleP* $\bigwedge P.\ P \Longrightarrow P \Longrightarrow P$, *rule A*, *rule A*)

end

context begin

private definition *bool-protect* (b::*bool*) ≡ b

lemma *bool-protectD*:
bool-protect $P \Longrightarrow P$
unfolding *bool-protect-def* **by** *simp*

lemma *bool-protectI*:
$P \Longrightarrow$ *bool-protect* P
unfolding *bool-protect-def* **by** *simp*

When you want to apply a rule/tactic to transform a potentially complex goal into another one manually, but want to indicate that any fresh emerging goals are solved by a more brutal method. E.g. apply (solves$_e merging frule$ x=... *in my-rulefastforce s*

method *solves-emerging* **methods** *m1 m2* = (*rule bool-protectD*, (*m1* ; (*rule bool-protectI* | (*m2*; *fail*))))

end

end

theory *Try-Methods*

imports *Eisbach-Methods*

keywords *trym* :: *diag*
and *add-try-method* :: *thy-decl*

begin

A collection of methods that can be "tried" against subgoals (similar to try, try0 etc). It is easy to add new methods with "add$_{t}ry_{m}ethod$", $althoughtheparsercurrentlysupportso$
Particular subgoals can be tried with "trym 1" etc. By default all subgoals are attempted unless they are coupled to others by shared schematic variables.

ML ‹

```
structure Try-Methods = struct

structure Methods = Theory-Data
(
  type T = Symtab.set;
  val empty = Symtab.empty;
  val extend = I;
  val merge = Symtab.merge (K true);
);

val get-methods-global = Methods.get #> Symtab.keys
val add-method = Methods.map o Symtab.insert-set

(* borrowed from try0 implementation (of course) *)
fun parse-method-name keywords =
  enclose ( )
  #> Token.explode keywords Position.start
  #> filter Token.is-proper
  #> Scan.read Token.stopper Method.parse
  #> (fn SOME (Method.Source src, -) => src | - => raise Fail expected Source);

fun mk-method ctxt = parse-method-name (Thy-Header.get-keywords' ctxt)
  #> Method.method-cmd ctxt
  #> Method.Basic

fun get-methods ctxt = get-methods-global (Proof-Context.theory-of ctxt)
  |> map (mk-method ctxt)

fun try-one-method m ctxt n goal
    = can (Timeout.apply (Time.fromSeconds 5)
        (Goal.restrict n 1 #> Method.NO-CONTEXT-TACTIC ctxt
            (Method.evaluate-runtime m ctxt []))
```

```
        #> Seq.hd
    )) goal

fun msg m-nm n = writeln (method  ^ m-nm ^  succceeded on goal  ^ string-of-int
n)

fun times xs ys = maps (fn x => map (pair x) ys) xs

fun independent-subgoals goal verbose = let
    fun get-vars t = Term.fold-aterms
        (fn (Var v) => Termtab.insert-set (Var v) | - => I)
        t Termtab.empty
    val goals = Thm.prems-of goal
    val goal-vars = map get-vars goals
    val count-vars = fold (fn t1 => fn t2 => Termtab.join (K (+))
        (Termtab.map (K (K 1)) t1, t2)) goal-vars Termtab.empty
    val indep-vars = Termtab.forall (fst #> Termtab.lookup count-vars
        #> (fn n => n = SOME 1))
    val indep = (1 upto Thm.nprems-of goal) ~~ map indep-vars goal-vars
    val - = app (fst #> string-of-int
        #> prefix ignoring non−independent goal  #> warning)
        (filter (fn x => verbose andalso not (snd x)) indep)
  in indep |> filter snd |> map fst end

fun try-methods opt-n ctxt goal = let
    val ms = get-methods-global (Proof-Context.theory-of ctxt)
        ~~ get-methods ctxt
    val ns = case opt-n of
        NONE => independent-subgoals goal true
      | SOME n => [n]
    fun apply ((m-nm, m), n) = if try-one-method m ctxt n goal
      then (msg m-nm n; SOME (m-nm, n)) else NONE
    val results = Par-List.map apply (times ms ns)
  in map-filter I results end

fun try-methods-command opt-n st = let
    val ctxt = #context (Proof.goal st)
        |> Try0.silence-methods false
    val goal = #goal (Proof.goal st)
  in try-methods opt-n ctxt goal; () end

val - = Outer-Syntax.command @{command-keyword trym}
  try methods from a library of specialised strategies
  (Scan.option Parse.int >> (fn opt-n =>
    Toplevel.keep-proof (try-methods-command opt-n o Toplevel.proof-of)))

fun local-check-add-method nm ctxt =
    (mk-method ctxt nm; Local-Theory.background-theory (add-method nm) ctxt)
```

val - = Outer-Syntax.command @{command-keyword add-try-method}
add a method to a library of strategies tried by trym
(Parse.name >> (Toplevel.local-theory NONE NONE o local-check-add-method))

end
⟩

add-try-method *fastforce*
add-try-method *blast*
add-try-method *metis*

method *auto-metis = solves ‹auto; metis›*
add-try-method *auto-metis*

end

theory *Extract-Conjunct*
imports
Main
Eisbach-Methods
begin

9 Extracting conjuncts in the conclusion

Methods for extracting a conjunct from a nest of conjuncts in the conclusion of a goal, typically by pattern matching.

When faced with a conclusion which is a big conjunction, it is often the case that a small number of conjuncts require special attention, while the rest can be solved easily by *clarsimp*, *auto* or similar. However, sometimes the method that would solve the bulk of the conjuncts would put some of the conjuncts into a more difficult or unsolvable state.

The higher-order methods defined here provide an efficient way to select a conjunct requiring special treatment, so that it can be dealt with first. Once all such conjuncts have been removed, the remaining conjuncts can all be solved together by some automated method.

Each method takes an inner method as an argument, and selects the left-most conjunct for which that inner method succeeds. The methods differ according to what they do with the selected conjunct. See below for more information and some simple examples.

context begin

9.1 Focused conjunct with context

We define a predicate which allows us to identify a particular sub-tree and its context within a nest of conjunctions. We express this sub-tree-with-context using a function which reconstructs the original nest of conjunctions. The context consists of a list of parent contexts, where each parent context consists of a sibling sub-tree, and a tag indicating whether the focused sub-tree is on the left or right. Rebuilding the original tree works from the focused sub-tree up towards the root of the original structure. This sub-tree-with-context is sometimes known as a zipper.

private fun *focus-conj* :: *bool* ⇒ *bool list* ⇒ *bool* **where**
  focus-conj current [] = *current*
| *focus-conj current* (*sibling* # *parents*) = *focus-conj* (*current* ∧ *sibling*) *parents*

private definition *focus* ≡ *focus-conj*

private definition *tag t P* ≡ *P*
private lemmas *focus-defs* = *focus-def tag-def*

private abbreviation *left* ≡ *tag Left*
private abbreviation *right* ≡ *tag Right*

private lemma *focus-example*:
  focus C [*right B*, *left D*, *left E*, *right A*] ⟷ *A* ∧ ((*B* ∧ *C*) ∧ *D*) ∧ *E*
  unfolding *focus-defs* **by** *auto*

9.2 Moving the focus

We now prove some rules which allow us to switch between focused and unfocused structures, and to move the focus around. Some versions of these rules carry an extra conjunct E outside the structure. Once we find the conjunct we want, this E allows to keep track of it while we reassemble the rest of the original structure.

First, we have rules for going between focused and unfocused structures.

private lemma *focus-top-iff*: *E* ∧ *focus P* [] ⟷ *E* ∧ *P*
  unfolding *focus-def* **by** *simp*

private lemmas *to-focus* = *focus-top-iff* [**where** *E*=*True*, *simplified*, *THEN iffD1*]
private lemmas *from-focusE* = *focus-top-iff* [*THEN iffD2*]
private lemmas *from-focus* = *from-focusE*[**where** *E*=*True*, *simplified*]

Next, we have rules for moving the focus to and from the left conjunct.

private lemma *focus-left-iff*: *E* ∧ *focus L* (*left R* # *P*) ⟷ *E* ∧ *focus* (*L* ∧ *R*) *P*
  unfolding *focus-defs* **by** *simp*

private lemmas *focus-left* = *focus-left-iff* [**where** E=*True*, *simplified*, *THEN iffD1*]
private lemmas *unfocusE-left* = *focus-left-iff* [*THEN iffD2*]
private lemmas *unfocus-left* = *unfocusE-left*[**where** E=*True*, *simplified*]

Next, we have rules for moving the focus to and from the right conjunct.

private lemma *focus-right-iff*: $E \wedge$ *focus* R (*right* L # P) $\longleftrightarrow$ $E \wedge$ *focus* ($L \wedge R$) P
unfolding *focus-defs* **using** *conj-commute* **by** *simp*

private lemmas *focus-right* = *focus-right-iff* [**where** E=*True*, *simplified*, *THEN iffD1*]
private lemmas *unfocusE-right* = *focus-right-iff* [*THEN iffD2*]
private lemmas *unfocus-right* = *unfocusE-right*[**where** E=*True*, *simplified*]

Finally, we have rules for extracting the current focus. The sibling of the extracted focus becomes the new focus of the remaining structure.

private lemma *extract-focus-iff*: *focus* C (*tag* t S # P) $\longleftrightarrow$ ($C \wedge$ *focus* S P)
unfolding *focus-defs* **by** (*induct* P *arbitrary*: S) *auto*

private lemmas *extract-focus* = *extract-focus-iff* [*THEN iffD2*]

9.3 Primitive methods for navigating a conjunction

Using these rules as transitions, we implement a machine which navigates a tree of conjunctions, searching from left to right for a conjunct for which a given method will succeed. Once a matching conjunct is found, it is extracted, and the remaining conjuncts are reassembled.

From the current focus, move to the leftmost sub-conjunct.

private method *focus-leftmost* = (*intro focus-left*)*?*

Find the furthest ancestor for which the current focus is still on the right.

private method *unfocus-rightmost* = (*intro unfocus-right*)*?*

Move to the immediate-right sibling.

private method *focus-right-sibling* = (*rule unfocus-left*, *rule focus-right*)

Move to the next conjunct in right-to-left ordering.

private method *focus-next-conjunct* = (*unfocus-rightmost*, *focus-right-sibling*, *focus-leftmost*)

Search from current focus toward the right until we find a matching conjunct.

private method *find-match* **methods** m = (*rule extract-focus*, m | *focus-next-conjunct*, *find-match* m)

Search within nest of conjuncts, leaving remaining structure focused.

private method *extract-match* **methods** m = (*rule to-focus*, *focus-leftmost*, *find-match* m)

Move all the way out of focus, keeping track of any extracted conjunct.

private method *unfocusE* = ((*intro unfocusE-right unfocusE-left*) *?*, *rule from-focusE*)
private method *unfocus* = ((*intro unfocus-right unfocus-left*) *?*, *rule from-focus*)

9.4 Methods for selecting the leftmost matching conjunct

See the introduction at the top of this theory for motivation, and below for some simple examples.

Assuming the conclusion of the goal is a nest of conjunctions, method *lift-conjunct* finds the leftmost conjunct for which the given method succeeds, and moves it to the front of the conjunction in the goal.

method *lift-conjunct* **methods** m = (*extract-match* ‹*succeeds* ‹*rule conjI*, m››, *unfocusE*)

Method *extract-conjunct* finds the leftmost conjunct for which the given method succeeds, and splits it into a fresh subgoal, leaving the remaining conjuncts untouched in the second subgoal. It is equivalent to *lift-conjunct* followed by *rule* $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$.

method *extract-conjunct* **methods** m = (*extract-match* ‹*rule conjI*, *succeeds* m›; *unfocus?*)

Method *apply-conjunct* finds the leftmost conjunct for which the given method succeeds, leaving any subgoals created by the application of that method, and a subgoal containing the remaining conjuncts untouched. It is equivalent to *extract-conjunct* followed by the given method, but more efficient.

method *apply-conjunct* **methods** m = (*extract-match* ‹*rule conjI*, m›; *unfocus?*)

9.5 Examples

Given an inner method based on *match*, which only succeeds on the desired conjunct C, *lift-conjunct* moves the conjunct C to the front. The body of the *match* here is irrelevant, since *lift-conjunct* always discards the effect of the method it is given.

lemma $[\![\ A;\ B;\ [\![\ A;\ B;\ D;\ E\ ]\!] \Longrightarrow C;\ D;\ E\ ]\!] \Longrightarrow A \wedge ((B \wedge C) \wedge D) \wedge E$
  apply (*lift-conjunct* ‹*match conclusion in* $C \Rightarrow$ ‹−››)
  — C as been moved to the front of the conclusion.
  apply (*match* **conclusion in** ‹$C \wedge A \wedge (B \wedge D) \wedge E$› $\Rightarrow$ ‹−›)
  oops

Method *extract-conjunct* works similarly, but peels of the matched conjunct as a separate subgoal. As for *lift-conjunct*, the effect of the given method is discarded, so the body of the *match* is irrelevant.

lemma $[\![\ A;\ B;\ [\![\ A;\ B;\ D;\ E\ ]\!] \Longrightarrow C;\ D;\ E\ ]\!] \Longrightarrow A \wedge ((B \wedge C) \wedge D) \wedge E$
  apply (*extract-conjunct* ‹*match conclusion in* $C \Rightarrow$ ‹−››)

— *extract-conjunct* gives us the matched conjunct C as a separate subgoal.
apply (*match* **conclusion in** $C \Rightarrow \langle - \rangle$)
apply *blast*
— The other subgoal contains the remaining conjuncts untouched.
apply (*match* **conclusion in** $\langle A \wedge (B \wedge D) \wedge E \rangle \Rightarrow \langle - \rangle$)
oops

Method *apply-conjunct* goes one step further, and applies the given method to the extracted subgoal.

lemma $[\![A; B; [\![A; B; D; E]\!] \Longrightarrow C; D; E]\!] \Longrightarrow A \wedge ((B \wedge C) \wedge D) \wedge E$
apply (*apply-conjunct* ⟨*match conclusion in* $C \Rightarrow$ ⟨*match premises in* H: - $\Rightarrow$ ⟨*rule* H⟩⟩⟩)
— We get four subgoals from applying the given method to the matched conjunct C.
apply (*match* **premises in** H: $A \Rightarrow \langle$*rule* $H\rangle$)
apply (*match* **premises in** H: $B \Rightarrow \langle$*rule* $H\rangle$)
apply (*match* **premises in** H: $D \Rightarrow \langle$*rule* $H\rangle$)
apply (*match* **premises in** H: $E \Rightarrow \langle$*rule* $H\rangle$)
— The last subgoal contains the remaining conjuncts untouched.
apply (*match* **conclusion in** $\langle A \wedge (B \wedge D) \wedge E \rangle \Rightarrow \langle - \rangle$)
oops

end

end

theory *Eval-Bool*

imports *Try-Methods*

begin

The eval$_bool method/simproc uses the code generator setup to reduce terms of boolean type to True or False$ *equations.*

Additional simprocs exist to reduce other types.

ML ⟨
structure Eval-Simproc = struct

exception Failure

fun mk-constname-tab ts = fold Term.add-const-names ts []
|> Symtab.make-set

fun is-built-from tab t = case Term.strip-comb t of
(Const (cn, -), ts) => Symtab.defined tab cn
andalso forall (is-built-from tab) ts
| - => false

```
fun eval tab ctxt ct = let
    val t = Thm.term-of ct
    val - = Term.fold-aterms (fn Free - => raise Failure
      | Var - => raise Failure | - => ignore) t ()
    val - = not (is-built-from tab t) orelse raise Failure
    val ev = the (try (Code-Simp.dynamic-conv ctxt) ct)
  in if is-built-from tab (Thm.term-of (Thm.rhs-of ev))
    then SOME ev else NONE end
  handle Failure => NONE | Option => NONE

val eval-bool = eval (mk-constname-tab [@{term True}, @{term False}])
val eval-nat = eval (mk-constname-tab [@{term Suc 0}, @{term Suc 1},
    @{term Suc 9}])
val eval-int = eval (mk-constname-tab [@{term 0 :: int}, @{term 1 :: int},
    @{term 18 :: int}, @{term (−9) :: int}])

val eval-bool-simproc = Simplifier.make-simproc @{context} eval-bool
  { lhss = [@{term b :: bool}], proc = K eval-bool }
val eval-nat-simproc = Simplifier.make-simproc @{context} eval-nat
  { lhss = [@{term n :: nat}], proc = K eval-nat }
val eval-int-simproc = Simplifier.make-simproc @{context} eval-int
  { lhss = [@{term i :: int}], proc = K eval-int }

end
›
```

method-setup *eval-bool* = ‹*Scan.succeed (fn ctxt => SIMPLE-METHOD′*
(CHANGED o full-simp-tac (clear-simpset ctxt
addsimprocs [Eval-Simproc.eval-bool-simproc])))›
use code generator setup to simplify booleans in goals to True or False

method-setup *eval-int-nat* = ‹*Scan.succeed (fn ctxt => SIMPLE-METHOD′*
(CHANGED o full-simp-tac (clear-simpset ctxt
addsimprocs [Eval-Simproc.eval-nat-simproc, Eval-Simproc.eval-int-simproc])))›
use code generator setup to simplify nats and ints in goals to values

add-try-method *eval-bool*

Testing.

definition
eval-bool-test-seq :: *int list*
where
eval-bool-test-seq = [*2*, *3*, *4*, *5*, *6*, *7*, *8*]

lemma
eval-bool-test-seq ! *4* = *6* ∧ (*3* :: *nat*) < *4*
∧ *sorted eval-bool-test-seq*
by *eval-bool*

A related gadget for installing constant definitions from locales as code equations. Useful where locales are being used to "hide" constants from the global state rather than to do anything tricky with interpretations.

Installing the global definitions in this way will allow eval$_booletcto"seethrough"thehidinganddecidequ$

ML ‹
structure Add-Locale-Code-Defs = struct

fun get-const-defs thy nm = Sign.consts-of thy
  |> Consts.dest |> #constants
  |> map fst
  |> filter (fn s => case Long-Name.explode s of
      [-, nm', -] => nm' = nm | - => false)
  |> map-filter (try (suffix -def #> Global-Theory.get-thm thy))
  |> filter (Thm.strip-shyps #> Thm.shyps-of #> null)
  |> tap (fn xs => tracing (Installing ^ string-of-int (length xs) ^ code defs))

fun setup nm thy = fold (fn t => Code.add-eqn-global (t, true))
  (get-const-defs thy nm) thy

end
›

locale *eval-bool-test-locale* **begin**

definition
  $x == (12 :: int)$

definition
  $y == (13 :: int)$

definition
  $z = (x * y) + x + y$

end

setup ‹*Add-Locale-Code-Defs.setup eval-bool-test-locale*›

setup ‹*Add-Locale-Code-Defs.setup eval-bool-test-locale*›

lemma *eval-bool-test-locale.z > 150*
  by *eval-bool*

end

— MLUtils is a collection of 'basic' ML utilities (kind of like `~~/src/Pure/library.ML`, but maintained by Trustworthy Systems). If you find yourself implementing: - A simple data-structure-shuffling task, - Something that shows up in the standard library of other functional languages, or - Something that's "missing" from the general pattern of an Isabelle ML library, consider adding it here.

theory *MLUtils*
imports *Main*
begin
ML-file *StringExtras.ML*
ML-file *ListExtras.ML*
ML-file *MethodExtras.ML*
ML-file *OptionExtras.ML*
ML-file *ThmExtras.ML*
ML-file *Sum.ML*
end

theory *Apply-Trace*
imports
  Main
  ml−helpers/MLUtils
begin

ML ‹
```
signature APPLY-TRACE =
sig
  val apply-results :
    {silent-fail : bool} ->
    (Proof.context -> thm -> ((string * int option) * term) list -> unit) ->
    Method.text-range -> Proof.state -> Proof.state Seq.result Seq.seq

  (* Lower level interface. *)
  val can-clear : theory -> bool
  val clear-deps : thm -> thm
  val join-deps : thm -> thm -> thm
  val used-facts : Proof.context -> thm -> ((string * int option) * term) list
  val pretty-deps: bool -> (string * Position.T) option -> Proof.context -> thm
->
    ((string * int option) * term) list -> Pretty.T
end

structure Apply-Trace : APPLY-TRACE =
struct

(*TODO: Add more robust oracle without hyp clearing *)
fun thm-to-cterm keep-hyps thm =
let

  val thy = Thm.theory-of-thm thm
```

```
  val pairs = Thm.tpairs-of thm
  val ceqs = map (Thm.global-cterm-of thy o Logic.mk-equals) pairs
  val hyps = Thm.chyps-of thm
  val prop = Thm.cprop-of thm
  val thm' = if keep-hyps then Drule.list-implies (hyps,prop) else prop

in
  Drule.list-implies (ceqs,thm') end


val (-, clear-thm-deps') =
 Context.>>> (Context.map-theory-result (Thm.add-oracle (Binding.name count-cheat,
thm-to-cterm false)));

fun clear-deps thm =
let

  val thm' = try clear-thm-deps' thm
 |> Option.map (fold (fn - => fn t => (@{thm Pure.reflexive} RS t)) (Thm.tpairs-of
thm))

in case thm' of SOME thm' => thm' | NONE => error Can't clear deps here end


fun can-clear thy = Context.subthy(@{theory},thy)

fun join-deps pre-thm post-thm =
let
  val pre-thm' = Thm.flexflex-rule NONE pre-thm |> Seq.hd
    |> Thm.adjust-maxidx-thm (Thm.maxidx-of post-thm + 1)
in
  Conjunction.intr pre-thm' post-thm |> Conjunction.elim |> snd
end

fun get-ref-from-nm' nm =
let
  val exploded = space-explode - nm;
  val base = List.take (exploded, (length exploded) - 1) |> space-implode -
  val idx = List.last exploded |> Int.fromString;
in if is-some idx andalso base <>  then SOME (base, the idx) else NONE end

fun get-ref-from-nm nm = Option.join (try get-ref-from-nm' nm);

fun maybe-nth l = try (curry List.nth l)

fun fact-from-derivation ctxt xnm =
let

  val facts = Proof-Context.facts-of ctxt;
```

```
  (* TODO: Check that exported local fact is equivalent to external one *)

  val idx-result =
    let
      val (name', idx) = get-ref-from-nm xnm |> the;
        val entry = try (Facts.retrieve (Context.Proof ctxt) facts) (name', Position.none) |> the;
      val thm = maybe-nth (#thms entry) (idx - 1) |> the;
    in SOME (xnm, thm) end handle Option => NONE;

  fun non-idx-result () =
    let
        val entry = try (Facts.retrieve (Context.Proof ctxt) facts) (xnm, Position.none) |> the;
      val thm = try the-single (#thms entry) |> the;
    in SOME (#name entry, thm) end handle Option => NONE;

in
  case idx-result of
    SOME thm => SOME thm
  | NONE => non-idx-result ()
end

fun most-local-fact-of ctxt xnm =
let
 val local-name = try (fn xnm => Long-Name.explode xnm |> tl |> tl |> Long-Name.implode) xnm |> the;
in SOME (fact-from-derivation ctxt local-name |> the) end handle Option =>
  fact-from-derivation ctxt xnm;

fun thms-of (PBody {thms,...}) = thms

fun proof-body-descend' f get-fact (ident, thm-node) deptab = let
  val nm = Proofterm.thm-node-name thm-node
  val body = Proofterm.thm-node-body thm-node
in
  (if not (f nm) then
     (Inttab.update-new (ident, SOME (nm, get-fact nm |> the)) deptab handle Inttab.DUP - => deptab)
  else raise Option) handle Option =>
    ((fold (proof-body-descend' f get-fact) (thms-of (Future.join body))
     (Inttab.update-new (ident, NONE) deptab)) handle Inttab.DUP - => deptab)
end

fun used-facts' f get-fact thm =
  let
    val body = thms-of (Thm.proof-body-of thm);

  in fold (proof-body-descend' f get-fact) body Inttab.empty end
```

```
fun used-pbody-facts ctxt thm =
  let
    val nm = Thm.get-name-hint thm;
    val get-fact = most-local-fact-of ctxt;
  in
    used-facts′ (fn nm′ => nm′ =  orelse nm′ = nm) get-fact thm
    |> Inttab.dest |> map-filter snd |> map snd |> map (apsnd (Thm.prop-of))
  end

fun raw-primitive-text f = Method.Basic (fn - => ((K (fn (ctxt, thm) => Se-
q.make-results (Seq.single (ctxt, f thm))))))


(∗Find local facts from new hyps∗)
fun used-local-facts ctxt thm =
let
  val hyps = Thm.hyps-of thm
  val facts = Proof-Context.facts-of ctxt |> Facts.dest-static true []

  fun match-hyp hyp =
  let
    fun get (nm,thms) =
      case (get-index (fn t => if (Thm.prop-of t) aconv hyp then SOME hyp else
NONE) thms)
      of SOME t => SOME (nm,t)
        | NONE => NONE


  in
    get-first get facts
  end

in
  map-filter match-hyp hyps end

fun used-facts ctxt thm =
  let
   val used-from-pbody = used-pbody-facts ctxt thm |> map (fn (nm,t) => ((nm,NONE),t))
      val used-from-hyps = used-local-facts ctxt thm |> map (fn (nm,(i,t)) =>
((nm,SOME i),t))
  in
    (used-from-hyps @ used-from-pbody)
  end

(∗ Perform refinement step, and run the given stateful function
   against computed dependencies afterwards. ∗)
fun refine args f text state =
let
```

```
  val ctxt = Proof.context-of state

  val thm = Proof.simple-goal state |> #goal

  fun save-deps deps = f ctxt thm deps


in
  if (can-clear (Proof.theory-of state)) then
     Proof.refine (Method.Combinator (Method.no-combinator-info,Method.Then,
[raw-primitive-text (clear-deps),text,
     raw-primitive-text (fn thm' => (save-deps (used-facts ctxt thm');join-deps thm
thm'))])) state
  else
     (if (#silent-fail args) then (save-deps [];Proof.refine text state) else error
Apply-Trace theory must be imported to trace applies)
end

(* Boilerplate from Proof.ML *)


fun method-error kind pos state =
  Seq.single (Proof-Display.method-error kind pos (Proof.raw-goal state));

fun apply args f text = Proof.assert-backward #> refine args f text #>
 Seq.maps-results (Proof.apply ((raw-primitive-text I),(Position.none, Position.none)));

fun apply-results args f (text, range) =
  Seq.APPEND (apply args f text, method-error (Position.range-position range));


structure Filter-Thms = Named-Thms
(
  val name = @{binding no-trace}
  val description = thms to be ignored from tracing
)

(* Print out the found dependencies. *)
fun pretty-deps only-names query ctxt thm deps =
let
  (* Remove duplicates. *)
 val deps = sort-distinct (prod-ord (prod-ord string-ord (option-ord int-ord)) Term-Ord.term-ord)
deps

  (* Fetch canonical names and theorems. *)
  val deps = map (fn (ident, term) => ThmExtras.adjust-thm-name ctxt ident
term) deps
```

```
  (* Remove boring theorems. *)
  val deps = subtract (fn (a, ThmExtras.FoundName (-, thm)) => Thm.eq-thm
(thm, a)
                        | - => false) (Filter-Thms.get ctxt) deps

  val deps = case query of SOME (raw-query,pos) =>
    let
      val pos' = perhaps (try (Position.advance-offsets 1)) pos;
      val q = Find-Theorems.read-query pos' raw-query;
       val results = Find-Theorems.find-theorems-cmd ctxt (SOME thm) (SOME
1000000000) false q
                   |> snd
                   |> map ThmExtras.fact-ref-to-name;

      (* Only consider theorems from our query. *)

     val deps = inter (fn (ThmExtras.FoundName (nmidx,-), ThmExtras.FoundName
(nmidx',-)) => nmidx = nmidx'
                                  | - => false) results deps
     in deps end
     | - => deps

in
  if only-names then
    Pretty.block
      (Pretty.separate  (map (ThmExtras.pretty-fact only-names ctxt) deps))
  else
  (* Pretty−print resulting theorems. *)
    Pretty.big-list used theorems:
      (map (Pretty.item o single o ThmExtras.pretty-fact only-names ctxt) deps)

end

val - = Context.>> (Context.map-theory Filter-Thms.setup)

end
⟩
```

end

theory *Apply-Trace-Cmd*
imports *Apply-Trace*
keywords *apply-trace* :: *prf-script*
begin

ML⟨

val - =
  Outer-Syntax.command @{*command-keyword apply-trace*} *initial refinement step* (*unstructured*)

  (*Args.mode only-names* −− (*Scan.option* (*Parse.position Parse.cartouche*)) −− *Method.parse* >>
    (*fn* ((*on*,*query*),*text*) => *Toplevel.proofs* (*Apply-Trace.apply-results* {*silent-fail* = *false*}
      (*Pretty.writeln ooo* (*Apply-Trace.pretty-deps on query*)) *text*)));

⟩

lemmas [*no-trace*] = *protectI protectD TrueI Eq-TrueI eq-reflection*

lemma $(a \wedge b) = (b \wedge a)$
  apply-trace *auto*
  oops

lemma $(a \wedge b) = (b \wedge a)$
  apply-trace ⟨*intro*⟩ *auto*
  oops

lemma
  assumes X: $b = a$
  assumes Y: $b = a$
  shows
  $b = a$
  apply-trace (*rule* Y)
  oops

locale *Apply-Trace-foo* = **fixes** b a
  assumes X: $b = a$
begin

  lemma shows $b = a$ $b = a$
   apply −
   apply-trace (*rule Apply-Trace-foo.X*)
   prefer *2*
   apply-trace (*rule* X)
   oops
end

experiment begin

Example of trace for grouped lemmas

definition *ex* :: *nat set* **where**
ex = {*1*,*2*,*3*,*4*}

lemma *v1*: *1* ∈ *ex* **by** (*simp add*: *ex-def*)
lemma *v2*: *2* ∈ *ex* **by** (*simp add*: *ex-def*)
lemma *v3*: *3* ∈ *ex* **by** (*simp add*: *ex-def*)

Group several lemmas in a single one

lemmas *vs* = *v1 v2 v3*

lemma *2* ∈ *ex*
  apply-trace (*simp add*: *vs*)
  oops

end
end

theory *Apply-Debug*
  imports
    Apply-Trace
    HOL−Eisbach.Eisbach-Tools
  keywords
    apply-debug :: *prf-script* % *proof* **and**
    continue :: *prf-script* % *proof* **and** *finish* :: *prf-script* % *proof*
begin

ML ‹
val start-max-threads = *Multithreading.max-threads* ();
›

context
begin

private method *put-prems* =
  (*match* **premises in** *H*:*PROP* - (*multi*) ⇒ ‹*insert H*›)

ML ‹
fun get-match-prems ctxt =
  let

    val st = *Goal.init* @{*cterm PROP P*}

    fun get-wrapped () =
      let
        val ((-,*st′*),-) =

Method-Closure.apply-method ctxt @{*method put-prems*} [] [] [] *ctxt* [] (*ctxt*, *st*)
|> *Seq.first-result prems*;

val prems =
Thm.prems-of st' |> *hd* |> *Logic.strip-imp-prems*;

in prems end

val match-prems = *the-default* [] (*try get-wrapped* ());

val all-prems = *Assumption.all-prems-of ctxt*;

in map-filter (*fn t* => *find-first* (*fn thm* => *t aconv* (*Thm.prop-of thm*)) *all-prems*) *match-prems end*

⟩
end

ML ⟨
signature APPLY-DEBUG =
sig
type break-opts = { *tags* : *string list*, *trace* : (*string* ∗ *Position.T*) *option*, *show-running* : *bool* }

val break : *Proof.context* −> *string option* −> *tactic*;
val apply-debug : *break-opts* −> *Method.text-range* −> *Proof.state* −> *Proof.state*;
val continue : *int option* −> (*context-state* −> *context-state option*) *option* −> *Proof.state* −> *Proof.state*;
val finish : *Proof.state* −> *Proof.state*;

val pretty-state: *Toplevel.state* −> *Pretty.T option*;

end

structure Apply-Debug : *APPLY-DEBUG* =
struct
type break-opts = { *tags* : *string list*, *trace* : (*string* ∗ *Position.T*) *option*, *show-running* : *bool* }

fun do-markup range m = *Output.report* [*Markup.markup* (*Markup.properties* (*Position.properties-of-range range*) *m*)];
fun do-markup-pos pos m = *Output.report* [*Markup.markup* (*Markup.properties* (*Position.properties-of pos*) *m*)];

type markup-queue = { *cur* : *Position.range option*, *next* : *Position.range option*, *clear-cur* : *bool* }

fun map-cur f ({*cur*, *next*, *clear-cur*} : *markup-queue*) =

```
  ({cur = f cur, next = next, clear-cur = clear-cur} : markup-queue)

fun map-next f ({cur, next, clear-cur} : markup-queue) =
  ({cur = cur, next = f next, clear-cur = clear-cur} : markup-queue)

fun map-clear-cur f ({cur, next, clear-cur} : markup-queue) =
  ({cur = cur, next = next, clear-cur = f clear-cur} : markup-queue)

type markup-state =
  { running : markup-queue
  }

fun map-running f ({running} : markup-state) =
  {running = f running}


structure Markup-Data = Proof-Data
(
  type T = markup-state Synchronized.var option *
    Position.range option (* latest method location *) *
    Position.range option (* latest breakpoint location *)
  fun init - : T = (NONE, NONE, NONE)
);

val init-queue = ({cur = NONE, next = NONE, clear-cur = false}: markup-queue)
val init-markup-state = ({running = init-queue} : markup-state)

fun set-markup-state id = Markup-Data.map (@{apply 3 (1)} (K id));
fun get-markup-id ctxt = #1 (Markup-Data.get ctxt);

fun set-latest-range range = Markup-Data.map (@{apply 3 (2)} (K (SOME range)));
fun get-latest-range ctxt = #2 (Markup-Data.get ctxt);

fun set-breakpoint-range range = Markup-Data.map (@{apply 3 (3)} (K (SOME
range)));
fun get-breakpoint-range ctxt = #3 (Markup-Data.get ctxt);

val clear-ranges = Markup-Data.map (@{apply 3 (3)} (K NONE) o @{apply 3
(2)} (K NONE));

fun swap-markup queue startm endm =
if is-some (#next queue) andalso #next queue = #cur queue then SOME (map-next
(K NONE) queue) else
let
  fun clear-cur () =
    (case #cur queue of SOME crng =>
        do-markup crng endm
      | NONE => ())
in
```

```
  case #next queue of SOME rng =>
     (clear-cur (); do-markup rng startm; SOME ((map-cur (K (SOME rng)) o
map-next (K NONE)) queue))
    | NONE => if #clear-cur queue then (clear-cur (); SOME ((map-cur (K
NONE) o map-clear-cur (K false)) queue))
            else NONE
end

fun markup-worker (SOME (id : markup-state Synchronized.var)) =
let
  fun main-loop () =
    let val - = Synchronized.guarded-access id (fn e =>
    case swap-markup (#running e) Markup.running Markup.finished of
      SOME queue' => SOME ((),map-running (fn - => queue') e)
    | NONE => NONE)
     in main-loop () end
in main-loop () end
 | markup-worker NONE = (fn () => ())

fun set-gen get set (SOME id) rng =
  let
    val - =
       Synchronized.guarded-access id (fn e =>
         if is-some (#next (get e)) orelse (#clear-cur (get e)) then NONE else
         if (#cur (get e)) = SOME rng then SOME ((), e)
         else (SOME ((),(set (map-next (fn - => SOME rng)) e))))

     val - = Synchronized.guarded-access id (fn e => if is-some (#next (get e))
then NONE else SOME ((),e))
  in () end
 | set-gen - - NONE - = ()


fun clear-gen get set (SOME id) =
  Synchronized.guarded-access id (fn e =>
  if (#clear-cur (get e)) then NONE
  else (SOME ((),(set (map-clear-cur (fn - => true)) e))))
 | clear-gen - - NONE = ()

val set-running = set-gen #running map-running
val clear-running = clear-gen #running map-running


fun traceify-method static-ctxt src =
let
  val range = Token.range-of src;
  val head-range = Token.range-of [hd src];
  val m = Method.method-cmd static-ctxt src;
```

```
in (fn eval-ctxt => fn facts =>
  let
    val eval-ctxt = set-latest-range head-range eval-ctxt;
    val markup-id = get-markup-id eval-ctxt;

    fun traceify seq = Seq.make (fn () =>
        let
          val - = set-running markup-id range;
          val r = Seq.pull seq;
          val - = clear-running markup-id;
        in Option.map (apsnd traceify) r end)

    fun tac (runtime-ctxt,thm) =
        let
          val runtime-ctxt' = set-latest-range head-range runtime-ctxt;
          val - = set-running markup-id range;
          in traceify (m eval-ctxt facts (runtime-ctxt', thm)) end

   in tac end)
end

fun add-debug ctxt (Method.Source src) = (Method.Basic (traceify-method ctxt sr-
c))
  | add-debug ctxt (Method.Combinator (x,y,txts)) = (Method.Combinator (x,y,
map (add-debug ctxt) txts))
  | add-debug - x = x

fun st-eq (ctxt : Proof.context,st) (ctxt',st') =
  pointer-eq (ctxt,ctxt') andalso Thm.eq-thm (st,st')

type result =
  { pre-state : thm,
    post-state : thm,
    context: Proof.context}

datatype final-state = RESULT of (Proof.context * thm) | ERR of (unit ->
string)

type debug-state =
  {results : result list, (* this execution, in order of appearance *)
   prev-results : thm list, (* continuations needed to get thread back to some state*)
   next-state : thm option, (* proof thread blocks waiting for this *)
    break-state : (Proof.context * thm) option, (* state of proof thread just before
blocking *)
   restart : (unit -> unit) * int, (* restart function (how many previous results to
keep), restart requested if non-zero *)
   final : final-state option, (* final result, maybe error *)
   trans-id : int, (* increment on every restart *)
   ignore-breaks: bool}
```

```
val init-state =
  ({results = [],
    prev-results = [],
    next-state = NONE, break-state = NONE,
    final = NONE, ignore-breaks = false, restart = (K (), ~1), trans-id = 0} :
debug-state)

fun map-next-state f ({results, next-state, break-state, final, ignore-breaks, prev-results,
restart, trans-id} : debug-state) =
  ({results = results, next-state = f next-state, break-state = break-state, final =
final, prev-results = prev-results,
   restart = restart, ignore-breaks = ignore-breaks, trans-id = trans-id} : debug-state)

fun map-results f ({results, next-state, break-state, final, ignore-breaks, prev-results,
restart, trans-id} : debug-state) =
  ({results = f results, next-state = next-state, break-state = break-state, final =
final, prev-results = prev-results,
   restart = restart, ignore-breaks = ignore-breaks, trans-id = trans-id} : debug-state)

fun map-prev-results f ({results, next-state, break-state, final, ignore-breaks, prev-results,
restart, trans-id} : debug-state) =
  ({results = results, next-state = next-state, break-state = break-state, final =
final, prev-results = f prev-results,
   restart = restart, ignore-breaks = ignore-breaks, trans-id = trans-id} : debug-state)

fun map-ignore-breaks f ({results, next-state, break-state = break-state, final, ignore-breaks,
prev-results, restart, trans-id} : debug-state) =
 ({results = results, next-state = next-state, break-state = break-state,final = final,
prev-results = prev-results,
   restart = restart, ignore-breaks = f ignore-breaks, trans-id = trans-id} : debug-state)

fun map-final f ({results, next-state, break-state, final, ignore-breaks, prev-results,
restart, trans-id} : debug-state) =
  ({results = results, next-state = next-state, break-state =break-state ,final = f
final, prev-results = prev-results,
   restart = restart, ignore-breaks = ignore-breaks, trans-id = trans-id} : debug-state)

fun map-restart f ({results, next-state, break-state, final, ignore-breaks, prev-results,
restart, trans-id} : debug-state) =
  ({results = results, next-state = next-state, break-state = break-state, final =
final, prev-results = prev-results,
   restart = f restart, ignore-breaks = ignore-breaks, trans-id = trans-id} : debug-state)

fun map-break-state f ({results, next-state, break-state, final, ignore-breaks, prev-results,
restart, trans-id} : debug-state) =
  ({results = results, next-state = next-state, break-state = f break-state, final =
final, prev-results = prev-results,
   restart = restart, ignore-breaks = ignore-breaks, trans-id = trans-id} : debug-state)
```

```
fun map-trans-id f ({results, next-state, break-state, final, ignore-breaks, prev-results,
restart, trans-id} : debug-state) =
  ({results = results, next-state = next-state, break-state = break-state, final =
final, prev-results = prev-results,
  restart = restart, ignore-breaks = ignore-breaks, trans-id = f trans-id} : debug-state)

fun is-restarting ({restart,...} : debug-state) = snd restart > ~1;
fun is-finished ({final,...} : debug-state) = is-some final;

val drop-states = map-break-state (K NONE) o map-next-state (K NONE);

fun add-result ctxt pre post = map-results (cons {pre-state = pre, post-state =
post, context = ctxt}) o drop-states;

fun get-trans-id (id : debug-state Synchronized.var) = #trans-id (Synchronized.value
id);

fun stale-transaction-err trans-id trans-id' =
  error (Stale transaction. Expected  ^ Int.toString trans-id ^  but found  ^ In-
t.toString trans-id')

fun assert-trans-id trans-id (e : debug-state) =
  if trans-id = (#trans-id e) then ()
    else stale-transaction-err trans-id (#trans-id e)

fun guarded-access id f =
  let
    val trans-id = get-trans-id id;
  in
  Synchronized.guarded-access id
    (fn (e : debug-state) =>
     (assert-trans-id trans-id e;
       (case f e of
          NONE => NONE
        | SOME (e', g) => SOME (e', g e))))
   end

fun guarded-read id f =
  let
    val trans-id = get-trans-id id;
  in
  Synchronized.guarded-access id
    (fn (e : debug-state) =>
     (assert-trans-id trans-id e;
      (case f e of
          NONE => NONE
        | SOME e' => SOME (e', e))))
   end
```

(∗ *Immediate return if there are previous results available or we are ignoring breakpoints* ∗)

```
fun pop-state-no-block id ctxt pre = guarded-access id (fn e =>
  if is-finished e then error Attempted to pop state from finished proof else
  if (#ignore-breaks e) then SOME (SOME pre, add-result ctxt pre pre) else
  case #prev-results e of
     [] => SOME (NONE, I)
   | (st :: sts) => SOME (SOME st, add-result ctxt pre st o map-prev-results (fn
- => sts)))

fun pop-next-state id ctxt pre = guarded-access id (fn e =>
  if is-finished e then error Attempted to pop state from finished proof else
  if not (null (#prev-results e)) then error Attempted to pop state when previous
results exist else
    if (#ignore-breaks e) then SOME (pre, add-result ctxt pre pre) else
    (case #next-state e of
           NONE => NONE
         | SOME st => SOME (st, add-result ctxt pre st)))

fun set-next-state id trans-id st = guarded-access id (fn e =>
  (assert-trans-id trans-id e;
   (if is-none (#next-state e) andalso is-some (#break-state e) then
      SOME ((), map-next-state (fn - => SOME st) o map-break-state (fn - =>
NONE))
    else error (Attempted to set next state in inconsistent state ^ (@{make-string}
e)))))

fun set-break-state id st = guarded-access id (fn e =>
  if is-none (#next-state e) andalso is-none (#break-state e) then
    SOME ((), map-break-state (fn - => SOME st))
  else error (Attempted to set break state in inconsistent state ^ (@{make-string}
e)))

fun pop-state id ctxt pre =
  case pop-state-no-block id ctxt pre of SOME st => st
  | NONE =>
  let
    val - = set-break-state id (ctxt, pre); (* wait for continue *)
  in pop-next-state id ctxt pre end

(* block until a breakpoint is hit or method finishes *)
fun wait-break-state id trans-id = guarded-read id
  (fn e =>
    (assert-trans-id trans-id e;
     (case (#final e) of SOME st => SOME (st, true) | NONE =>
```

```
      case (#break-state e) of SOME st => SOME (RESULT st, false)
     | NONE => NONE)));

fun debug-print (id : debug-state Synchronized.var) =
  (@{print} (Synchronized.value id));

(* Trigger a restart if an existing nth entry differs from the given one *)
fun maybe-restart id n st =
let
  val gen = guarded-read id (fn e => SOME (#trans-id e));

  val did-restart = guarded-access id (fn e =>
    if is-some (#next-state e) then NONE else
    if not (null (#prev-results e)) then NONE else
    if is-restarting e then NONE (* TODO, what to do if we're already restarting?
*)
    else if length (#results e) > n then
      (SOME (true, map-restart (apsnd (fn - => n))))
    else SOME (false, I))


  val trans-id = Synchronized.guarded-access id
    (fn e => if is-restarting e then NONE else
             if not did-restart orelse gen + 1 = #trans-id e then SOME (#trans-id
e,e) else
             stale-transaction-err (gen + 1) (#trans-id e));
in trans-id end;


fun peek-all-results id = guarded-read id (fn e => SOME (#results e));

fun peek-final-result id =
  guarded-read id (fn e => #final e)

fun poke-error (RESULT st) = st
  | poke-error (ERR e) = error (e ())

fun context-state e = (#context e, #pre-state e);

fun nth-pre-result id i = guarded-read id
  (fn e =>
      if length (#results e) > i then SOME (RESULT (context-state (nth (rev
(#results e)) i)), false) else
    if not (null (#prev-results e)) then NONE else
      (if length (#results e) = i then
         (case #break-state e of SOME st => SOME (RESULT st, false) | NONE
=> NONE) else
         (case #final e of SOME st => SOME (st, true) | NONE => NONE)))
```

```
fun set-finished-result id trans-id st =
  guarded-access id (fn e =>
  (assert-trans-id trans-id e;
   SOME ((), map-final (K (SOME st)))));

fun is-finished-result id = guarded-read id (fn e => SOME (is-finished e));

fun get-finish id =
if is-finished-result id then peek-final-result id else
  let
    val - = guarded-access id
      (fn - => SOME ((), (map-ignore-breaks (fn - => true))))

  in peek-final-result id end


val no-break-opts = ({tags = [], trace = NONE, show-running = false} : break-opts)

structure Debug-Data = Proof-Data
(
  type T = debug-state Synchronized.var option  (* handle on active proof thread
*) *
  int * (* continuation counter *)
  bool * (* currently interactive context *)
  break-opts * (* global break arguments *)
  string option (* latest breakpoint tag *)
  fun init - : T = (NONE,~1, false, no-break-opts, NONE)
);

fun set-debug-ident ident = Debug-Data.map  (@{apply 5 (1)} (fn - => SOME
ident))
val get-debug-ident = #1 o Debug-Data.get;
val get-the-debug-ident = the o get-debug-ident;

fun set-break-opts opts = Debug-Data.map (@{apply 5 (4)} (fn - => opts))
val get-break-opts = #4 o Debug-Data.get;

fun set-last-tag tags = Debug-Data.map (@{apply 5 (5)} (fn - => tags))
val get-last-tag = #5 o Debug-Data.get;

val is-debug-ctxt = is-some o #1 o Debug-Data.get;

fun clear-debug ctxt = ctxt
 |> Debug-Data.map (fn - => (NONE,~1,false, no-break-opts, NONE))
 |> clear-ranges


val get-continuation = #2 o Debug-Data.get;
val get-can-break = #3 o Debug-Data.get;
```

```
(* Maintain pointer equality if possible *)
fun set-continuation i ctxt = if get-continuation ctxt = i then ctxt else
  Debug-Data.map (@{apply 5 (2)} (fn - => i)) ctxt;

fun set-can-break b ctxt = if get-can-break ctxt = b then ctxt else
  Debug-Data.map (@{apply 5 (3)} (fn - => b)) ctxt;

fun has-break-tag (SOME tag) tags = member (=) tags tag
  | has-break-tag NONE - = true;

fun break ctxt tag = (fn thm =>
if not (get-can-break ctxt)
   orelse Method.detect-closure-state thm
   orelse not (has-break-tag tag (#tags (get-break-opts ctxt)))
    then Seq.single thm else
  let
    val id = get-the-debug-ident ctxt;
    val ctxt' = set-last-tag tag ctxt;

    val st' = Seq.make (fn () =>
     SOME (pop-state id ctxt' thm,Seq.empty))

  in st' end)

fun init-interactive ctxt = ctxt
  |> set-can-break false
  |> Config.put Method.closure true;

type static-info =
  {private-dyn-facts : string list, local-facts : (string * thm list) list}

structure Data = Generic-Data
(
  type T = (morphism * Proof.context * static-info) option;
  val empty: T = NONE;
  val extend = K NONE;
  fun merge data : T = NONE;
);

(* Present Eisbach/Match variable binding context as normal context elements.
   Potentially shadows existing facts/binds *)

fun dest-local s =
  let
    val [local,s'] = Long-Name.explode s;
  in SOME s' end handle Bind => NONE

fun maybe-bind st (-,[tok]) ctxt =
```

```
if Method.detect-closure-state st then
  let
    val target = Local-Theory.target-of ctxt
    val local-facts = Proof-Context.facts-of ctxt;
   val global-facts = map (Global-Theory.facts-of) (Context.parents-of (Proof-Context.theory-of
ctxt));
    val raw-facts = Facts.dest-all (Context.Proof ctxt) true global-facts local-facts
|> map fst;

    fun can-retrieve s = can (Facts.retrieve (Context.Proof ctxt) local-facts) (s,
Position.none)

    val private-dyns = raw-facts |>
      (filter (fn s => Facts.is-concealed local-facts s andalso Facts.is-dynamic
local-facts s
                 andalso can-retrieve (Long-Name.base-name s)
                 andalso Facts.intern local-facts (Long-Name.base-name s) = s
                 andalso not (can-retrieve s)) )

   val local-facts = Facts.dest-static true [(Proof-Context.facts-of target)] local-facts;

    val - = Token.assign (SOME (Token.Declaration (fn phi =>
     Data.put (SOME (phi,ctxt, {private-dyn-facts = private-dyns, local-facts =
local-facts})))) tok;

  in ctxt end
else
  let
    val SOME (Token.Declaration decl) = Token.get-value tok;
    val dummy-ctxt = decl Morphism.identity (Context.Proof ctxt);
   val SOME (phi,static-ctxt,{private-dyn-facts, local-facts}) = Data.get dummy-ctxt;

    val old-facts = Proof-Context.facts-of static-ctxt;
    val cur-priv-facts = map (fn s =>
          Facts.retrieve (Context.Proof ctxt) old-facts (Long-Name.base-name
s,Position.none)) private-dyn-facts;

    val cur-local-facts =
      map (fn (s,fact) => (dest-local s, Morphism.fact phi fact)) local-facts
     |> map-filter (fn (s,fact) => case s of SOME s => SOME (s,fact) | - =>
NONE)

    val old-fixes = (Variable.dest-fixes static-ctxt)

    val local-fixes =
      filter (fn (-,f) =>
        Variable.is-newly-fixed static-ctxt (Local-Theory.target-of static-ctxt) f)
old-fixes
      |> map-filter (fn (n,f) => case Variable.default-type static-ctxt f of SOME
```

```
typ =>
            if typ = dummyT then NONE else SOME (n, Free (f, typ))
          | NONE => NONE)

      val local-binds = (map (apsnd (Morphism.term phi)) local-fixes)

      val ctxt' = ctxt
      |> fold (fn (s,t) =>
          Variable.bind-term ((s,0),t)
        #> Variable.declare-constraints (Var ((s,0),Term.fastype-of t))) local-binds
      |> fold (fn e =>
           Proof-Context.put-thms true (Long-Name.base-name (#name e), SOME
(#thms e))) cur-priv-facts
      |> fold (fn (nm,fact) =>
          Proof-Context.put-thms true (nm, SOME fact)) cur-local-facts
    |> Proof-Context.put-thms true (match-prems, SOME (get-match-prems ctxt));

    in ctxt' end
 | maybe-bind - - ctxt = ctxt

val - = Context.>> (Context.map-theory (Method.setup @{binding #}
 (Scan.lift (Scan.trace (Scan.trace (Args.$$$ break) -- (Scan.option Parse.string)))
>>
   (fn ((b,tag),toks) => fn - => fn - =>
    fn (ctxt,thm) =>
      (let

        val range = Token.range-of toks;
        val ctxt' = ctxt
          |> maybe-bind thm b
          |> set-breakpoint-range range;

       in Seq.make-results (Seq.map (fn thm' => (ctxt',thm')) (break ctxt' tag thm))
end))) ))

fun map-state f state =
      let
       val (r,-) = Seq.first-result map-state (Proof.apply
         (Method.Basic (fn - => fn - => fn st =>
             Seq.make-results (Seq.single (f st))),
           Position.no-range) state)
      in r end;

fun get-state state =
let
  val {context,goal} = Proof.simple-goal state;
in (context,goal) end
```

```
fun maybe-trace (SOME (tr, pos)) (ctxt, st) =
let
  val deps = Apply-Trace.used-facts ctxt st;
  val query = if tr =  then NONE else SOME (tr, pos);
  val pr = Apply-Trace.pretty-deps false query ctxt st deps;
in Pretty.writeln pr end
  | maybe-trace NONE (ctxt, st) = ()

val active-debug-threads = Synchronized.var active-debug-threads ([] : unit future
list);

fun update-max-threads extra =
let
  val n-active = Synchronized.change-result active-debug-threads (fn ts =>
    let
      val ts' = List.filter (not o Future.is-finished) ts;
    in (length ts',ts') end)
  val - = Multithreading.max-threads-update (start-max-threads + ((n-active + ex-
tra) * 3));
in () end


fun continue i-opt m-opt =
(map-state (fn (ctxt,thm) =>
      let

        val ctxt = set-can-break true ctxt

        val thm = Apply-Trace.clear-deps thm;

          val - = if is-none (get-debug-ident ctxt) then error Cannot continue in a
non−debug state else ();

        val id = get-the-debug-ident ctxt;

        val start-cont = get-continuation ctxt; (* how many breakpoints so far *)

        val trans-id = maybe-restart id start-cont (ctxt,thm);
          (* possibly restart if the thread has made too much progress.
             trans-id is the current number of restarts, used to avoid manipulating
             stale states *)

         val - = nth-pre-result id start-cont; (* block until we've hit the start of this
continuation *)

        fun get-final n (st as (ctxt,-))  =
         case (i-opt,m-opt) of
            (SOME i,NONE) => if i < 1 then error Can only continue a positive
number of breakpoints else
```

```
    if n = start-cont + i then SOME st else NONE
  | (NONE, SOME m) => (m (apfst init-interactive st))
  | (-, -) => error Invalid continue arguments

  val ex-results = peek-all-results id |> rev;

  fun tick-up n (-,thm) =
    if n < length ex-results then error Unexpected number of existing results
      (*case get-final n (#pre-state (nth ex-results n)) of SOME st' => (st',
false, n)
      | NONE => tick-up (n + 1) st *)
    else
    let
     val - = if n > length ex-results then set-next-state id trans-id thm else ();
      val (n-r, b) = wait-break-state id trans-id;
      val st' = poke-error n-r;
    in if b then (st',b, n) else
      case get-final n st' of SOME st'' => (st'', false, n)
      | NONE => tick-up (n + 1) st' end

  val - = if length ex-results < start-cont then
 (debug-print id; @{print} (start-cont,start-cont); @{print} (trans-id,trans-id);
      error Unexpected number of existing results)
    else ()


  val (st',b, cont) = tick-up (start-cont + 1) (ctxt, thm)

  val st'' = if b then (Output.writeln Final Result.; st' |> apfst clear-debug)
           else st' |> apfst (set-continuation cont) |> apfst (init-interactive);

  (* markup for matching breakpoints to continues *)

  val sr = serial ();

  fun markup-def rng =
    (Output.report
       [Markup.markup (Markup.entity breakpoint
        |> Markup.properties (Position.entity-properties-of true sr
             (Position.range-position rng))) ]);

  val - = Option.map markup-def (get-latest-range (fst st''));
  val - = Option.map markup-def (get-breakpoint-range (fst st''));

  val - =
    (Context-Position.report ctxt (Position.thread-data ())
       (Markup.entity breakpoint
            |> Markup.properties (Position.entity-properties-of false sr Posi-
tion.none)))
```

```
    val - = maybe-trace (#trace (get-break-opts ctxt)) st'';

   in st'' end))

fun do-apply pos rng opts m =
let
  val {tags, trace, show-running} = opts;
  val batch-mode = is-some (Position.line-of (fst rng));
  val show-running = if batch-mode then false else show-running;

  val - = if batch-mode then () else update-max-threads 1;

in
 (fn st => map-state (fn (ctxt,thm) =>
  let
     val ident = Synchronized.var debug-state init-state;
     val markup-id = if show-running then SOME (Synchronized.var markup-state
init-markup-state)
       else NONE;
     fun maybe-markup m = if show-running then do-markup rng m else ();

     val - = if is-debug-ctxt ctxt then
      error Cannot use apply-debug while debugging else ();

     val m = apfst (fn f => f ctxt) m;

     val st = Proof.map-context
      (set-can-break true
       #> set-break-opts opts
       #> set-markup-state markup-id
       #> set-debug-ident ident
       #> set-continuation ~1) st
       |> map-state (apsnd Apply-Trace.clear-deps);

     fun do-cancel thread = (Future.cancel thread; Future.join-result thread; ());

     fun do-fork trans-id = Future.fork (fn () =>
       let
        val (ctxt,thm) = get-state st;

        val r = case Exn.interruptible-capture (fn st =>
        let val - = Seq.pull (break ctxt NONE thm) in
        (case (Seq.pull o Proof.apply m) st
          of (SOME (Seq.Result st', -)) => RESULT (get-state st')
           | (SOME (Seq.Error e, -)) => ERR e
           | - => ERR (fn - => No results)) end) st
          of Exn.Res (RESULT r) => RESULT r
            | Exn.Res (ERR e) => ERR e
```

```
      | Exn.Exn e => ERR (fn _ => Runtime.exn_message e)
    val _ = set_finished_result ident trans_id r;

    val _ = clear_running markup_id;

   in () end)


  val thread = do_fork 0;
  val _ = Synchronized.change ident (map_restart (fn _ => (fn () => do_cancel
thread, ~1)));

  val _ = maybe_markup Markup.finished;

  val _ = Future.fork (fn () => markup_worker markup_id ());

 val st' = get_state (continue (SOME 1) NONE (Proof.map_context (set_continuation
0) st))

  val _ = maybe_markup Markup.joined;


 val main_thread = if batch_mode then Future.fork (fn () => ()) else Future.fork
(fn () =>
  let

   fun restart_state gls e = e
     |> map_prev_results (fn _ => map #post_state (take gls (rev (#results
e))))
    |> map_results (fn _ => [])
    |> map_final (fn _ => NONE)
    |> map_ignore_breaks (fn _ => false)
    |> map_restart (fn _ => (K (), gls))
    |> map_break_state (fn _ => NONE)
    |> map_next_state (fn _ => NONE)
    |> map_trans_id (fn i => i + 1);

   fun main_loop () =
    let
     val r = Synchronized.timed_access ident (fn _ => SOME (seconds 0.1))
(fn e as {restart,next_state,...} =>
      if is_restarting e andalso is_none next_state then
       SOME ((fst restart, #trans_id e), restart_state (snd restart) e) else
NONE);
     val _ = OS.Process.sleep (seconds 0.1);
     in case r of NONE => main_loop ()
     | SOME (f,trans_id) =>
      let
       val _ = f ();
```

```
                val - = clear-running markup-id;
                val thread = do-fork (trans-id + 1);
                 val - = Synchronized.change ident (map-restart (fn - => (fn () =>
do-cancel thread, ~1)))
              in main-loop () end
           end;
       in main-loop () end);

       val - = maybe-markup Markup.running;
       val - = maybe-markup Markup.forked;

       val - = Synchronized.change active-debug-threads (cons main-thread);

   in st' end) st)
end

fun apply-debug opts (m', rng) =
  let
      val - = Method.report (m', rng);

      val m'' = (fn ctxt => add-debug ctxt m')
      val m = (m'',rng)
      val pos = Position.thread-data ();

     in do-apply pos rng opts m end;

fun quasi-keyword x = Scan.trace (Args.$$$ x) >>
  (fn (s,[tok]) => (Position.reports [(Token.pos-of tok, Markup.quasi-keyword)];
s))

val parse-tags = (Args.parens (quasi-keyword tags |-- Parse.enum1 , Parse.string));
val parse-trace = Scan.option (Args.parens (quasi-keyword trace |-- Scan.option
(Parse.position Parse.cartouche))) >>
  (fn SOME NONE => SOME (, Position.none) | SOME (SOME x) => SOME
x | - => NONE);

val parse-opts1 = (parse-tags -- parse-trace) >>
  (fn (tags,trace) => {tags = tags, trace = trace});

val parse-opts2 = (parse-trace -- (Scan.optional parse-tags [])) >>
  (fn (trace,tags) => {tags = tags, trace = trace});

fun mode s = Scan.optional (Args.parens (quasi-keyword s) >> (K true)) false

val parse-opts = ((parse-opts1 || parse-opts2) -- mode show-running) >>
  (fn ({tags, trace}, show-running) => {tags = tags, trace = trace, show-running
= show-running} : break-opts) ;

val - =
```

Outer-Syntax.command @{*command-keyword apply-debug*} *initial goal refinement step* (*unstructured*)
(*Scan.trace*
(*parse-opts* −− *Method.parse*) >>
(*fn* ((*opts*, (*m*,-)),*toks*) => *Toplevel.proof* (*apply-debug opts* (*m*,*Token.range-of toks*))));

val finish = *map-state* (*fn* (*ctxt*,-) =>
let
val - = *if is-none* (*get-debug-ident ctxt*) *then error Cannot finish in a non*−*debug state else* ();
val f = *get-finish* (*get-the-debug-ident ctxt*);
in f |> *poke-error* |> *apfst clear-debug end*)

fun continue-cmd i-opt m-opt state =
let
val {*context*,...} = *Proof.simple-goal state*;
val check = *Method.map-source* (*Method.method-closure* (*init-interactive context*))

val m-opt′ = *Option.map* (*check o Method.check-text context o fst*) *m-opt*;

fun eval-method txt =
(*fn* (*ctxt*,*thm*) => *try* (*fst o Seq.first-result method*) (*Method.evaluate txt ctxt* [] (*ctxt*,*thm*)))

val i-opt′ = *case* (*i-opt*,*m-opt*) *of* (*NONE*,*NONE*) => *SOME 1* | - => *i-opt*;

in continue i-opt′ (*Option.map eval-method m-opt′*) *state end*

val - =
Outer-Syntax.command @{*command-keyword continue*} *step to next breakpoint*
(*Scan.option Parse.int* −− *Scan.option Method.parse* >> (*fn* (*i-opt*,*m-opt*) =>
(*Toplevel.proof* (*continue-cmd i-opt m-opt*))))

val - =
Outer-Syntax.command @{*command-keyword finish*} *finish debugging*
(*Scan.succeed* (*Toplevel.proof* (*continue NONE* (*SOME* (*fn* - => *NONE*)))))

fun pretty-hidden-goals ctxt0 thm =
let
val ctxt = *ctxt0*
|> *Config.put show-types* (*Config.get ctxt0 show-types orelse Config.get ctxt0 show-sorts*)
|> *Config.put show-sorts false*;

```
    val prt-term =
      singleton (Syntax.uncheck-terms ctxt) #>
      Type-Annotation.ignore-free-types #>
      Syntax.unparse-term ctxt;
    val prt-subgoal = prt-term

    fun pretty-subgoal s A =
      Pretty.markup (Markup.subgoal s) [Pretty.str (  ^ s ^ . ), prt-subgoal A];
    fun pretty-subgoals n = map-index (fn (i, A) => pretty-subgoal (string-of-int
(i + n)) A);

    fun collect-extras prop =
      case try Logic.unprotect prop of
      SOME prop' =>
      (if Logic.count-prems prop' > 0 then
        (case try Logic.strip-horn prop'
            of SOME (As, B) =>  As :: collect-extras B
            | NONE => [])
      else [])
      | NONE => []

     val (As,B) = Logic.strip-horn (Thm.prop-of thm);
     val extras' = collect-extras B;
      val extra-goals-limit = Int.max (Config.get ctxt0 Goal-Display.goals-limit −
length As, 0);
     val all-extras = flat (take (length extras' − 1) extras');
     val extras = take extra-goals-limit all-extras;

     val pretty = pretty-subgoals (length As + 1) extras @
       (if extra-goals-limit < length all-extras then
            [Pretty.str (A total of  ^ (string-of-int (length all-extras)) ^  hidden
subgoals...)]
        else [])
  in pretty end

fun pretty-state state =
  if Toplevel.is-proof state
    then
  let
    val st = Toplevel.proof-of state;
    val {goal, context, ...} = Proof.raw-goal st;
    val pretty = Toplevel.pretty-state state;
    val hidden = pretty-hidden-goals context goal;
    val out = pretty @
      (if length hidden > 0 then [Pretty.keyword1 hidden goals] @ hidden else []);
  in SOME (Pretty.chunks out) end
  else NONE

end
```

⟩

ML ⟨*val - =*
```
  Query-Operation.register {name = print-state, pri = Task-Queue.urgent-pri}
    (fn {state = st, output-result, ...} =>
      case Apply-Debug.pretty-state st of
        SOME prt => output-result (Markup.markup Markup.state (Pretty.string-of prt))
      | NONE => ());⟩
```

end

theory *Find-Names*
imports *Pure*
keywords *find-names* :: *diag*
begin

The **find-names** command, when given a theorem, finds other names the theorem appears under, via matching on the whole proposition. It will not identify unnamed theorems.

ML ⟨

```
local
(* all-facts-of and pretty-ref taken verbatim from non−exposed version
   in Find-Theorems.ML of official Isabelle/HOL distribution *)
fun all-facts-of ctxt =
  let
    val thy = Proof-Context.theory-of ctxt;
    val transfer = Global-Theory.transfer-theories thy;
    val local-facts = Proof-Context.facts-of ctxt;
    val global-facts = Global-Theory.facts-of thy;
  in
   (Facts.dest-all (Context.Proof ctxt) false [global-facts] local-facts
    @ Facts.dest-all (Context.Proof ctxt) false [] global-facts)
   |> maps Facts.selections
   |> map (apsnd transfer)
  end;

fun pretty-ref ctxt thmref =
  let
    val (name, sel) =
      (case thmref of
        Facts.Named ((name, -), sel) => (name, sel)
      | Facts.Fact - => raise Fail Illegal literal fact);
  in
    [Pretty.marks-str (#1 (Proof-Context.markup-extern-fact ctxt name), name),
      Pretty.str (Facts.string-of-selection sel)]
  end;
```

in

fun find-names ctxt thm =
  let
    fun eq-filter body thmref = (*body* = *Thm.full-prop-of* (*snd thmref*));
  in
    (*filter* (*eq-filter* (*Thm.full-prop-of thm*))) (*all-facts-of ctxt*)
    |> *map #1*
  end;

fun pretty-find-names ctxt thm =
  let
    val results = *find-names ctxt thm*;
   val position-markup = *Position.markup* (*Position.thread-data* ()) *Markup.position*;
  in
    ((*Pretty.mark position-markup* (*Pretty.keyword1 find-names*)) ::
      Par-List.map (*Pretty.item o* (*pretty-ref ctxt*)) *results*)
    |> *Pretty.fbreaks* |> *Pretty.block* |> *Pretty.writeln*
  end

end

val - =
  Outer-Syntax.command @{*command-keyword find-names*}
    find other names of a named theorem
    (*Parse.thms1* >> (*fn srcs* => *Toplevel.keep* (*fn st* =>
      pretty-find-names (*Toplevel.context-of st*)
        (*hd* (*Attrib.eval-thms* (*Toplevel.context-of st*) *srcs*)))));
⟩

end

theory *TSubst*
imports
  Main
begin

method-setup *tsubst* = ⟨
  Scan.lift (*Args.mode asm* −−
            Scan.optional (*Args.parens* (*Scan.repeat Parse.nat*)) [*0*] −−
            Parse.term)
   >> (*fn* ((*asm*,*occs*),*t*) => (*fn ctxt* =>
   Method.SIMPLE-METHOD (*Subgoal.FOCUS-PARAMS* (*fn focus* => (*fn thm*
=>
  let

```
(* This code used to use Thm.certify-inst in 2014, which was removed.
   The following is just a best guess for what it did. *)
fun certify-inst ctxt (typ-insts, term-insts) =
     (typ-insts
      |> map (fn (tvar, inst) =>
            (Thm.ctyp-of ctxt (TVar tvar),
             Thm.ctyp-of ctxt inst)),
      term-insts
      |> map (fn (var, inst) =>
            (Thm.cterm-of ctxt (Var var),
             Thm.cterm-of ctxt inst)))

val ctxt' = #context focus

val ((-, schematic-terms), ctxt2) =
  Variable.import-inst true [(#concl focus) |> Thm.term-of] ctxt'
  |>> certify-inst ctxt'

  val ctxt3 = fold (fn (t,t') => Variable.bind-term (Thm.term-of t |> Term.dest-Var |> fst, (t' |> Thm.term-of))) schematic-terms ctxt2


val athm = Syntax.read-term ctxt3 t
     |> Object-Logic.ensure-propT ctxt'
     |> Thm.cterm-of ctxt'
     |> Thm.trivial

  val thm' = Thm.instantiate ([], map (apfst (Thm.term-of #> dest-Var)) schematic-terms) thm

in
  (if asm then EqSubst.eqsubst-asm-tac else EqSubst.eqsubst-tac)
    ctxt3 occs [athm] 1 thm'
    |> Seq.map (singleton (Variable.export ctxt3 ctxt'))
   end)) ctxt 1)))
⟩ subst, with term instead of theorem as equation
```

schematic-goal
  assumes a: $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y$
  fixes x :: $'b$
  shows $\bigwedge x$ ::$'a$ :: *type*. $?Q\ x \Longrightarrow P\ x \wedge ?Q\ x$
  apply (*tsubst* (*asm*) $?Q\ x = (P\ x \wedge P\ x)$)
   apply (*rule refl*)
  apply (*tsubst* $P\ x = P\ y$,*simp add*:a)+
  apply (*tsubst* (*2*) $P\ y = P\ x$, *simp add*:a)
  apply (*clarsimp simp*: a)
  done

end

theory *Time-Methods-Cmd* **imports**
Main
begin

ML ‹
```
structure Time-Methods = struct
  (* Work around Isabelle running every apply method on a dummy proof state *)
  fun skip-dummy-state (method: Method.method) : Method.method =
    fn facts => fn (ctxt, st) =>
      case Thm.prop-of st of
        Const (Pure.prop, -) $ (Const (Pure.term, -) $ Const (Pure.dummy-pattern,
-)) =>
          Seq.succeed (Seq.Result (ctxt, st))
        | - => method facts (ctxt, st);

 (* ML interface. Takes a list of (possibly−named) methods, then calls the supplied
  * callback with the method index (starting from 1), supplied name and timing.
  * Also returns the list of timings at the end. *)
  fun time-methods
        (no-check: bool)
        (skip-fail: bool)
        (callback: (int * string option −> Timing.timing −> unit))
        (maybe-named-methods: (string option * Method.method) list)
        (* like Method.method but also returns timing list *)
       : thm list −> context-state −> (Timing.timing list * context-state Seq.result
Seq.seq)
    = fn facts => fn (ctxt, st) => let
        fun run method =
              Timing.timing (fn () =>
                case method facts (ctxt, st) |> Seq.pull of
                  (* Peek at first result, then put it back *)
                    NONE => (NONE, Seq.empty)
                  | SOME (r as Seq.Result (-, st′), rs) => (SOME st′, Seq.cons r rs)
                  | SOME (r as Seq.Error -, rs) => (NONE, Seq.cons r rs)
              ) ()

        val results = tag-list 1 maybe-named-methods
              |> map (fn (idx1, (maybe-name, method)) =>
                  let val (time, (st′, results)) = run method
                      val - =
                        if Option.isSome st′ orelse not skip-fail
                        then callback (idx1, maybe-name) time
                        else ()
                    val name = Option.getOpt (maybe-name, [method ^ string-of-int
idx1 ^ ])
```

```
          in {name = name, state = st', results = results, time = time} end)

    val canonical-result = hd results
    val other-results = tl results
    val return-val = (map #time results, #results canonical-result)
    fun show-state NONE = @{thm FalseE[where P=METHOD-FAILED]}
      | show-state (SOME st) = st
  in
    if no-check then return-val else
    (* Compare the proof states that we peeked at *)
    case other-results
         |> filter (fn result =>
              (* It's tempting to use aconv, etc., here instead of (<>), but
               * minute differences such as bound names in Pure.all can
               * break a proof script later on. *)
              Option.map Thm.full-prop-of (#state result) <>
              Option.map Thm.full-prop-of (#state canonical-result)) of
        [] => return-val
      | (bad-result::-) =>
          raise THM (methods \ ^ #name canonical-result ^
                 \ and \ ^ #name bad-result ^ \ have different results,
                 1, map (show-state o #state) [canonical-result, bad-result])
  end
end
›

method-setup time-methods = ‹
let
 fun scan-flag name = Scan.lift (Scan.optional (Args.parens (Parse.reserved name)
>> K true) false)
  val parse-no-check = scan-flag no-check
  val parse-skip-fail = scan-flag skip-fail
 val parse-maybe-name = Scan.option (Scan.lift (Parse.liberal-name --| Parse.$$$
:))
  fun auto-name (idx1, maybe-name) =
      Option.getOpt (maybe-name, [method  ^ string-of-int idx1 ^ ])
in
  parse-no-check -- parse-skip-fail --
  Scan.repeat1 (parse-maybe-name -- Method.text-closure) >>
  (fn ((no-check, skip-fail), maybe-named-methods-text) => fn ctxt =>
    let
      val max-length = tag-list 1 (map fst maybe-named-methods-text)
                    |> map (String.size o auto-name)
                    |> (fn ls => fold (curry Int.max) ls 0)
      fun pad-name s =
         let val pad-length = max-length + String.size : - String.size s
         in s ^ replicate-string pad-length  end
        fun timing-callback id time = warning (pad-name (auto-name id ^ : ) ^
Timing.message time)
```

```
    val maybe-named-methods = maybe-named-methods-text
        |> map (apsnd (fn method-text => Method.evaluate method-text ctxt))
   val timed-method = Time-Methods.time-methods no-check skip-fail timing-callback
maybe-named-methods
    fun method-discard-times facts st = snd (timed-method facts st)
   in
    method-discard-times
    |> Time-Methods.skip-dummy-state
   end)
end
› Compare running time of several methods on the current proof state
```

end

theory *Try-Attribute*
imports *Main*
begin

```
ML ‹
local

val parse-warn = Scan.lift (Scan.optional (Args.parens (Parse.reserved warn) >>
K true) false)

val attribute-generic = Context.cases Attrib.attribute-global Attrib.attribute

fun try-attribute-cmd (warn, attr-srcs) (ctxt, thm) =
  let
    val attrs = map (attribute-generic ctxt) attr-srcs
    val (th′, context′) =
      fold (uncurry o Thm.apply-attribute) attrs (thm, ctxt)
      handle e =>
        (if Exn.is-interrupt e then Exn.reraise e
         else if warn then warning (TRY: ignoring exception:  ^ (@{make-string}
e))
         else ();
        (thm, ctxt))
  in (SOME context′, SOME th′) end

in

val - = Theory.setup
  (Attrib.setup @{binding TRY}
    (parse-warn -- Attrib.attribs >> try-attribute-cmd)
    higher order attribute combinator to try other attributes, ignoring failure)

end
›
```

The *TRY* attribute is an attribute combinator that applies other attributes, ignoring any failures by returning the original state. Note that since attributes are applied separately to each theorem in a theorem list, *TRY* will leave failing theorems unchanged while modifying the rest.

Accepts a "warn" flag to print any errors encountered.

Usage: thm foo[TRY [¡attributes¿]]

thm foo[TRY (warn) [¡attributes¿]]

10 Examples

experiment begin
  lemma *eq1*: (*1* :: *nat*) = *1* + *0* **by** *simp*
  lemma *eq2*: (*2* :: *nat*) = *1* + *1* **by** *simp*

  lemmas *eqs* = *eq1 TrueI eq2*

'eqs[symmetric]' would fail because there are no unifiers with *True*, but *TRY* ignores that.

  lemma
    1 + *0* = (*1* :: *nat*)
    True
    1 + *1* = (*2* :: *nat*)
    by (*rule eqs*[*TRY* [*symmetric*]])+

You can chain calls to *TRY* at the top level, to apply different attributes to different theorems.

  lemma *ineq*: (*1* :: *nat*) < *2* **by** *simp*
  lemmas *ineqs* = *eq1 ineq*
  lemma
    1 + *0* = (*1* :: *nat*)
    (*1* :: *nat*) ≤ *2*
    by (*rule ineqs*[*TRY* [*symmetric*], *TRY* [*THEN order.strict-implies-order*]])+

You can chain calls to *TRY* within each other, to chain more attributes onto particular theorems.

  lemmas *more-eqs* = *eq1 eq2*
  lemma
    1 = (*1* :: *nat*)
    1 + *1* = (*2* :: *nat*)
    by (*rule more-eqs*[*TRY* [*symmetric*, *TRY* [*simplified add-0-right*]]])+

The 'warn' flag will print out any exceptions encountered. Since *symmetric* doesn't apply to *True* or *1* < *2*, this will log two errors.

  lemmas *yet-another-group* = *eq1 TrueI eq2 ineq*
  thm *yet-another-group*[*TRY* (*warn*) [*symmetric*]]

TRY should handle pretty much anything it might encounter.

thm *eq1*[*TRY* (*warn*) [**where** *x*=*5*]]
thm *eq1*[*TRY* (*warn*) [*OF refl*]]
end

end

term$_pat : MLantiquotationforpatternmatchingonterms.$
See TermPatternAntiquote$_Testsforexamplesandtests.$

theory *TermPatternAntiquote* **imports**
Pure
begin

ML ‹
```
structure Term-Pattern-Antiquote = struct

val quote-string = quote

(* typ matching; doesn't support matching on named TVars.
 * This is because each TVar is likely to appear many times in the pattern. *)
fun gen-typ-pattern (TVar -) = -
  | gen-typ-pattern (TFree (v, sort)) =
      Term.TFree ( ^ quote-string v ^ , [ ^ commas (map quote-string sort) ^ ])
  | gen-typ-pattern (Type (typ-head, args)) =
      Term.Type ( ^ quote-string typ-head ^ , [ ^ commas (map gen-typ-pattern
args) ^ ])

(* term matching; does support matching on named (non−dummy) Vars.
 * The ML var generated will be identical to the Var name except in
 * indexed names like ?v1.2, which creates the var v12. *)
fun gen-term-pattern (Var ((-dummy-, -), -)) = -
  | gen-term-pattern (Var ((v, 0), -)) = v
  | gen-term-pattern (Var ((v, n), -)) = v ^ string-of-int n
  | gen-term-pattern (Const (n, typ)) =
      Term.Const ( ^ quote-string n ^ ,  ^ gen-typ-pattern typ ^ )
  | gen-term-pattern (Free (n, typ)) =
      Term.Free ( ^ quote-string n ^ ,  ^ gen-typ-pattern typ ^ )
  | gen-term-pattern (t as f $ x) =
      (* (read-term-pattern -) helpfully generates a dummy var that is
       * applied to all bound vars in scope. We go back and remove them. *)
      let fun default () = ( ^ gen-term-pattern f ^  $  ^ gen-term-pattern x ^ );
      in case strip-comb t of
            (h as Var ((-dummy-, -), -), bs) =>
              if forall is-Bound bs then gen-term-pattern h else default ()
          | - => default () end
  | gen-term-pattern (Abs (-, typ, t)) =
      Term.Abs (-,  ^ gen-typ-pattern typ ^ ,  ^ gen-term-pattern t ^ )
  | gen-term-pattern (Bound n) = Bound  ^ string-of-int n
```

(* *Create term pattern. All Var names must be distinct in order to generate ML variables.* *)
fun term-pattern-antiquote ctxt s =
  let val pat = *Proof-Context.read-term-pattern ctxt s*
    val add-var-names′ = *fold-aterms* (*fn Var* (*v*, -) => *curry* (::) *v* | - => *I*);
    val vars = *add-var-names′ pat* [] |> *filter* (*fn* (*n*, -) => *n* <> -*dummy*-)
    val - = *if vars* = *distinct* (=) *vars then* () *else*
        raise TERM (*Pattern contains duplicate vars*, [*pat*])
  in (ˆ *gen-term-pattern pat* ˆ) *end*

end;
val - = *Context*.>> (*Context.map-theory* (
  ML-Antiquotation.inline @{*binding term-pat*}
    ((*Args.context* −− *Scan.lift Args.embedded-inner-syntax*)
      >> *uncurry Term-Pattern-Antiquote.term-pattern-antiquote*)))
⟩

end

theory *Trace-Schematic-Insts*
imports
  Main
  ml−helpers/MLUtils
  ml−helpers/TermPatternAntiquote
begin

See Trace$_S chematic_I nsts_T estfortestsandexamples.$

locale *data-stash*
begin

We use this to stash a list of the schematics in the conclusion of the proof state. After running a method, we can read off the schematic instantiations (if any) from this list, then restore the original conclusion. Schematic types are added as "undefined :: ?'a" (for now, we don't worry about types that don't have sort "type").

TODO: there ought to be some standard way of stashing things into the proof state. Find out what that is and refactor

definition *container* :: *′a* ⇒ *bool* ⇒ *bool*
  where
  container a b ≡ *True*

lemma *proof-state-add*:
  Pure.prop PROP P ≡ *PROP Pure.prop* (*container True xs* ⟹ *PROP P*)
  by (*simp add*: *container-def*)

lemma *proof-state-remove*:
  PROP Pure.prop (*container True xs* ⟹ *PROP P*) ≡ *Pure.prop* (*PROP P*)

by (*simp add*: *container-def*)

lemma *rule-add*:
PROP P ≡ (*container True xs* ⟹ *PROP P*)
by (*simp add*: *container-def*)

lemma *rule-remove*:
(*container True xs* ⟹ *PROP P*) ≡ *PROP P*
by (*simp add*: *container-def*)

lemma *elim*:
container a b
by (*simp add*: *container-def*)

ML ‹

```
signature TRACE-SCHEMATIC-INSTS = sig
  type instantiations = (term * (int * term)) list * (typ * typ) list

  val trace-schematic-insts:
        Method.method -> (instantiations -> unit) -> Method.method
  val default-report:
        Proof.context -> string -> instantiations -> unit

  val trace-schematic-insts-tac:
        Proof.context ->
        (instantiations -> instantiations -> unit) ->
        (thm -> int -> tactic) ->
        thm -> int -> tactic
  val default-rule-report:
        Proof.context -> string -> instantiations -> instantiations -> unit

  val skip-dummy-state: Method.method -> Method.method
  val make-term-container: term list -> term
  val dest-term-container: term -> term list

  val attach-proof-annotations: Proof.context -> term list -> thm -> thm
  val detach-proof-annotations: Proof.context -> thm -> (int * term) list * thm

  val attach-rule-annotations: Proof.context -> term list -> thm -> thm
  val detach-rule-result-annotations: Proof.context -> thm -> (int * term) list *
thm
end

structure Trace-Schematic-Insts: TRACE-SCHEMATIC-INSTS = struct
```

— Each pair is a (schematic, instantiation) pair.
The int in the term instantiations is the number of binders which are due to subgoal bounds.
An explanation: if we instantiate some schematic '?P' within a subgoal like $\bigwedge x\ y.$

Q, it might be instantiated to $\lambda a.\ R\ a\ x$. We need to capture 'x' when reporting the instantiation, so we report that '?P' has been instantiated to $\lambda x\ y\ a.\ R\ a\ x$. In order to distinguish between the bound 'x', 'y', and 'a', we record that the two outermost binders are actually due to the subgoal bounds.

```
type instantiations = (term * (int * term)) list * (typ * typ) list

— Work around Isabelle running every apply method on a dummy proof state
fun skip-dummy-state method =
  fn facts => fn (ctxt, st) =>
    case Thm.prop-of st of
        Const (@{const-name Pure.prop}, -) $
       (Const (@{const-name Pure.term}, -) $ Const (@{const-name Pure.dummy-pattern},
-)) =>
          Seq.succeed (Seq.Result (ctxt, st))
      | - => method facts (ctxt, st);

— Utils
fun rewrite-state-concl eqn st =
  Conv.fconv-rule (Conv.concl-conv (Thm.nprems-of st) (K eqn)) st

— Strip the Pure.prop that wraps proof state conclusions
fun strip-prop ct =
      case Thm.term-of ct of
      Const (@{const-name Pure.prop}, @{typ prop ⇒ prop}) $ - => Thm.dest-arg
ct
      | - => raise CTERM (strip-prop: head is not Pure.prop, [ct])

fun cconcl-of st =
  funpow (Thm.nprems-of st) Thm.dest-arg (Thm.cprop-of st)
  |> strip-prop

fun vars-of-term t =
  Term.add-vars t []
  |> sort-distinct Term-Ord.var-ord

fun type-vars-of-term t =
  Term.add-tvars t []
  |> sort-distinct Term-Ord.tvar-ord

— Create annotation list
fun make-term-container ts =
      fold (fn t => fn container =>
              Const (@{const-name container},
                   fastype-of t --> @{typ bool ⇒ bool}) $
                 t $ container)
        (rev ts) @{term True}

— Retrieve annotation list
fun dest-term-container
```

```
    (Const (@{const-name container}, -) $ x $ list) =
        x :: dest-term-container list
  | dest-term-container - = []
```

— Attach some terms to a proof state, by "hiding" them in the protected goal.

```
fun attach-proof-annotations ctxt terms st =
  let
    val container = make-term-container terms
    (* FIXME: this might affect st's maxidx *)
    val add-eqn =
          Thm.instantiate
            ([],
             [(((P, 0), @{typ prop}), cconcl-of st),
              (((xs, 0), @{typ bool}), Thm.cterm-of ctxt container)])
            @{thm proof-state-add}
  in
    rewrite-state-concl add-eqn st
  end
```

— Retrieve attached terms from a proof state

```
fun detach-proof-annotations ctxt st =
  let
    val st-concl = cconcl-of st
    val (ccontainer', real-concl) = Thm.dest-implies st-concl
    val ccontainer =
          ccontainer'
          |> Thm.dest-arg (* strip Trueprop *)
          |> Thm.dest-arg — strip outer container True
    val terms =
          ccontainer
          |> Thm.term-of
          |> dest-term-container
    val remove-eqn =
          Thm.instantiate
            ([],
             [(((P, 0), @{typ prop}), real-concl),
              (((xs, 0), @{typ bool}), ccontainer)])
            @{thm proof-state-remove}
  in
    (map (pair 0) terms, rewrite-state-concl remove-eqn st)
  end
```

— Attaches the given terms to the given thm by stashing them as a new *container* premise, *after* all the existing premises (this minimises disruption when the rule is used with things like ‘erule‘).

```
fun attach-rule-annotations ctxt terms thm =
  let
    val container = make-term-container terms
    (* FIXME: this might affect thm's maxidx *)
```

```
    val add-eqn =
          Thm.instantiate
            ([],
             [(((P, 0), @{typ prop}), Thm.cconcl-of thm),
              (((xs, 0), @{typ bool}), Thm.cterm-of ctxt container)])
            @{thm rule-add}
  in
    rewrite-state-concl add-eqn thm
  end
```

— Finds all the variables and type variables in the given thm, then uses ‘attach‘ to stash them in a *container* within the thm.
Returns a tuple containing the variables and type variables which were attached this way.

```
fun annotate-with-vars-using (attach: Proof.context -> term list -> thm -> thm) ctxt thm =
  let
    val tvars = type-vars-of-term (Thm.prop-of thm) |> map TVar
    val tvar-carriers = map (fn tvar => Const (@{const-name undefined}, tvar)) tvars
    val vars = vars-of-term (Thm.prop-of thm) |> map Var
    val annotated-rule = attach ctxt (vars @ tvar-carriers) thm
  in ((vars, tvars), annotated-rule) end

val annotate-rule = annotate-with-vars-using attach-rule-annotations
val annotate-proof-state = annotate-with-vars-using attach-proof-annotations

fun split-and-zip-instantiations (vars, tvars) insts =
    let val (var-insts, tvar-insts) = chop (length vars) insts
    in (vars ~~ var-insts, tvars ~~ map (snd #> fastype-of) tvar-insts) end
```

— Term version of `Thm.dest_arg`.

```
val dest-arg = Term.dest-comb #> snd
```

— Cousin of `Term.strip_abs`.

```
fun strip-all t = (Term.strip-all-vars t, Term.strip-all-body t)
```

— Matches subgoals of the form:
$\bigwedge A\ B\ C.\ [\![X;\ Y;\ Z]\!] \Longrightarrow$ *container True data*
Extracts the instantiation variables from ‘?data‘, and re-applies the surrounding meta abstractions (in this case ‘And¿A B C‘).

```
fun dest-instantiation-container-subgoal t =
    let
      val (vars, goal) = t |> strip-all
      val goal = goal |> Logic.strip-imp-concl
    in
      case goal of
        @{term-pat Trueprop (container True ?data)} =>
            dest-term-container data
```

```
        |> map (fn t => (length vars, Logic.rlist-abs (rev vars, t))) (* reapply
variables *)
        |> SOME
    | - => NONE
  end
```

— Finds the first subgoal with a *container* conclusion. Extracts the data from the container and removes the subgoal.

```
fun detach-rule-result-annotations ctxt st =
 let
  val (idx, data) =
     st
     |> Thm.prems-of
     |> Library.get-index dest-instantiation-container-subgoal
     |> OptionExtras.get-or-else (fn () => error No container subgoal!)
  val st' =
     st
     |> resolve-tac ctxt @{thms elim} (idx + 1)
     |> Seq.hd
 in
  (data, st')
 end
```

— ‘abs$_a$*llnt‘wrapsthefirst‘n‘lambdaabstractionsin‘t‘withinterleavedPure.allconstructors.Forexample,‘abs*$_a$
ab.lambda > c.P”.Theresultingtermisusuallynotwell − typed.
Used to disambiguate schematic instantiations where the instantiation is a lambda.

```
fun abs-all 0 t = t
 | abs-all n (t as (Abs (v, typ, body))) =
    if n < 0 then error Number of lambdas to wrap should be positive. else
    Const (@{const-name Pure.all}, dummyT)
     $ Abs (v, typ, abs-all (n − 1) body)
 | abs-all n - = error (Expected at least  ^ Int.toString n ^  more lambdas.)

fun filtered-instantiation-lines ctxt (var-insts, tvar-insts) =
 let
  val vars-lines =
     map (fn (var, (abs, inst)) =>
       if var = inst then  (* don't show unchanged *) else
            ^ Syntax.string-of-term ctxt var ^   =>   ^
          Syntax.string-of-term ctxt (abs-all abs inst) ^ \n)
     var-insts
  val tvars-lines =
     map (fn (tvar, inst) =>
       if tvar = inst then  (* don't show unchanged *) else
            ^ Syntax.string-of-typ ctxt tvar ^   =>   ^
          Syntax.string-of-typ ctxt inst ^ \n)
     tvar-insts
 in
  vars-lines @ tvars-lines
```

```
  end
```

— Default callback for black-box method tracing. Prints nontrivial instantiations to tracing output with the given title line.

```
fun default-report ctxt title insts =
  let
    val all-insts = String.concat (filtered-instantiation-lines ctxt insts)
  (* TODO: add a quiet flag, to suppress output when nothing was instantiated *)
  in title ^ \n ^ (if all-insts =  then   (no instantiations)\n else all-insts)
     |> tracing
  end
```

— Default callback for tracing rule applications. Prints nontrivial instantiations to tracing output with the given title line. Separates instantiations of rule variables and goal variables.

```
fun default-rule-report ctxt title rule-insts proof-insts =
  let
    val rule-lines = String.concat (filtered-instantiation-lines ctxt rule-insts)
    val rule-lines =
        if rule-lines =
        then (no rule instantiations)\n
        else rule instantiations:\n ^ rule-lines;
    val proof-lines = String.concat (filtered-instantiation-lines ctxt proof-insts)
    val proof-lines =
        if proof-lines =
        then (no goal instantiations)\n
        else goal instantiations:\n ^ proof-lines;
  in title ^ \n ^ rule-lines ^ \n ^ proof-lines |> tracing  end
```

— ‘trace$_{s}chematic_{i}nsts_{t}acctxtcallbacktacticthmidx$‘$doesthefollowing:$
- Produce a *container*-annotated version of ‘thm‘. - Runs ‘tactic‘ on subgoal ‘idx‘, using the annotated version of ‘thm‘. - If the tactic succeeds, call ‘callback‘ with the rule instantiations and the goal instantiations, in that order.

```
fun trace-schematic-insts-tac
    ctxt
    (callback: instantiations -> instantiations -> unit)
    (tactic: thm -> int -> tactic)
    thm idx st =
  let
    val (rule-vars, annotated-rule) = annotate-rule ctxt thm
    val (proof-vars, annotated-proof-state) = annotate-proof-state ctxt st
    val st = tactic annotated-rule idx annotated-proof-state
  in
    st |> Seq.map (fn st =>
      let
        val (rule-terms, st) = detach-rule-result-annotations ctxt st
        val (proof-terms, st) = detach-proof-annotations ctxt st
        val rule-insts = split-and-zip-instantiations rule-vars rule-terms
        val proof-insts = split-and-zip-instantiations proof-vars proof-terms
```

```
      val () = callback rule-insts proof-insts
    in
      st
    end
  )
end
```

— ML interface, calls the supplied function with schematic unifications (will be given all variables, including those that haven't been instantiated).

```
fun trace-schematic-insts (method: Method.method) callback
  = fn facts => fn (ctxt, st) =>
    let
      val (vars, annotated-st) = annotate-proof-state ctxt st
    in (* Run the method *)
      method facts (ctxt, annotated-st)
      |> Seq.map-result (fn (ctxt', annotated-st') => let
            (* Retrieve the stashed list, now with unifications *)
            val (annotations, st') = detach-proof-annotations ctxt' annotated-st'
            val insts = split-and-zip-instantiations vars annotations
            (* Report the list *)
            val - = callback insts
         in (ctxt', st') end)
    end

end
›
```

end

method-setup *trace-schematic-insts* = ‹

```
  let
    open Trace-Schematic-Insts
  in
    (Scan.option (Scan.lift Parse.liberal-name) -- Method.text-closure) >>
    (fn (maybe-title, method-text) => fn ctxt =>
      trace-schematic-insts
          (Method.evaluate method-text ctxt)
          (default-report ctxt
              (Option.getOpt (maybe-title, trace-schematic-insts:)))
      |> skip-dummy-state
    )
  end
```

› *Method combinator to trace schematic variable and type instantiations*

end

theory *Insulin*

imports
Pure
keywords
desugar-term desugar-thm desugar-goal :: *diag*
begin

ML ‹

```
structure Insulin = struct

val desugar-random-tag = dsfjdssdfsd
fun fresh-substring s = let
  fun next [] = [#a]
    | next (#z :: n) = #a :: next n
    | next (c :: n) = Char.succ c :: n
  fun fresh n = let
    val ns = String.implode n
    in if String.isSubstring ns s then fresh (next n) else ns end
  in fresh [#a] end

(* Encode a (possibly qualified) constant name as an (expected−to−be−)unused
name.
 * The encoded name will be treated as a free variable. *)
fun escape-const c = let
  val delim = fresh-substring c
  in desugar-random-tag ^ delim ^ - ^
       String.concat (case Long-Name.explode c of
                        (a :: b :: xs) => a :: map (fn x => delim ^ x) (b :: xs)
                      | xs => xs)
  end
(* Decode; if it fails, return input string *)
fun unescape-const s =
  if not (String.isPrefix desugar-random-tag s) then s else
 let val cs = String.extract (s, String.size desugar-random-tag, NONE) |> String.explode
      fun readDelim d (#- :: cs) = (d, cs)
        | readDelim d (c :: cs) = readDelim (d @ [c]) cs
      val (delim, cs) = readDelim [] cs
      val delimlen = length delim
      fun splitDelim name cs =
            if take delimlen cs = delim then name :: splitDelim [] (drop delimlen cs)
              else case cs of [] => if null name then [] else [name]
                            | (c::cs) => splitDelim (name @ [c]) cs
      val names = splitDelim [] cs
  in Long-Name.implode (map String.implode names) end
  handle Match => s

fun dropQuotes s = if String.isPrefix \ s andalso String.isSuffix \ s
                     then String.substring (s, 1, String.size s − 2) else s
```

```
(* Translate markup from consts−encoded−as−free−variables to actual consts *)
fun desugar-reconst ctxt (tr as XML.Elem ((tag, attrs), children))
  = if tag = fixed orelse tag = intensify then
      let val s = XML.content-of [tr]
          val name = unescape-const s
          fun get-entity-attrs (XML.Elem ((entity, attrs), -)) = SOME attrs
            | get-entity-attrs (XML.Elem (-, body)) =
                find-first (K true) (List.mapPartial get-entity-attrs body)
            | get-entity-attrs (XML.Text -) = NONE
      in
        if name = s then tr else
          (* try to look up the const's info *)
          case Syntax.read-term ctxt name
               |> Thm.cterm-of ctxt
               |> Proof-Display.pp-cterm (fn - => Proof-Context.theory-of ctxt)
               |> Pretty.string-of
               |> dropQuotes
               |> YXML.parse
               |> get-entity-attrs of
              SOME attrs =>
                XML.Elem ((entity, attrs), [XML.Text name])
            | - =>
                XML.Elem ((entity, [(name, name), (kind, constant)]),
                          [XML.Text name]) end
    else XML.Elem ((tag, attrs), map (desugar-reconst ctxt) children)
  | desugar-reconst - (t as XML.Text -) = t

fun term-to-string ctxt no-markup =
  Syntax.pretty-term ctxt
  #> Pretty.string-of
  #> YXML.parse-body
  #> map (desugar-reconst ctxt)
  #> (if no-markup then XML.content-of else YXML.string-of-body)
  #> dropQuotes

(* Strip constant names from a term.
 * A term is split to a term-unconst and a string list of the
 * const names in tree preorder. *)
datatype term-unconst =
    UCConst of typ |
    UCAbs of string * typ * term-unconst |
    UCApp of term-unconst * term-unconst |
    UCVar of term

fun is-ident-char c = Char.isAlphaNum c orelse c = #- orelse c = #. orelse c =
#'

fun term-to-unconst (Const (name, typ)) =
```

```
    (* some magical constants have strange names, such as ==>; ignore them *)
      if forall is-ident-char (String.explode name) then (UCConst typ, [name])
        else (UCVar (Const (name, typ)), [])
  | term-to-unconst (Abs (var, typ, body)) = let
      val (body', consts) = term-to-unconst body
      in (UCAbs (var, typ, body'), consts) end
  | term-to-unconst (f $ x) = let
      val (f', consts1) = term-to-unconst f
      val (x', consts2) = term-to-unconst x
      in (UCApp (f', x'), consts1 @ consts2) end
  | term-to-unconst t = (UCVar t, [])

fun term-from-unconst (UCConst typ) (name :: consts) =
      ((if unescape-const name = name then Const else Free) (name, typ), consts)
  | term-from-unconst (UCAbs (var, typ, body)) consts = let
      val (body', consts) = term-from-unconst body consts
      in (Abs (var, typ, body'), consts) end
  | term-from-unconst (UCApp (f, x)) consts = let
      val (f', consts) = term-from-unconst f consts
      val (x', consts) = term-from-unconst x consts
      in (f' $ x', consts) end
  | term-from-unconst (UCVar v) consts = (v, consts)

(* Count occurrences of bad strings.
 * Bad strings are allowed to overlap, but for each string, non−overlapping occur-
rences are counted.
 * Note that we search on string lists, to deal with symbols correctly. *)
fun count-matches (haystack: ''a list) (needles: ''a list list): int list =
  let (* Naive algorithm. Probably ok, given that we're calling the term printer a
lot elsewhere. *)
      fun try-match xs [] = SOME xs
        | try-match (x::xs) (y::ys) = if x = y then try-match xs ys else NONE
        | try-match - - = NONE
      fun count [] = 0
        | count needle = let
            fun f [] occs = occs
              | f haystack' occs = case try-match haystack' needle of
                                     NONE => f (tl haystack') occs
                                   | SOME tail => f tail (occs + 1)
            in f haystack 0 end
  in map count needles end

fun focus-list (xs: 'a list): ('a list * 'a * 'a list) list =
  let fun f head x [] = [(head, x, [])]
        | f head x (tail as x'::tail') = (head, x, tail) :: f (head @ [x]) x' tail'
  in case xs of [] => []
              | (x::xs) => f [] x xs end

(* Do one rewrite pass: try every constant in sequence, then collect the ones which
```

```
 * reduced the occurrences of bad strings *)
fun rewrite-pass ctxt (t: term) (improved: term -> bool) (escape-const: string ->
string): term =
  let val (ucterm, consts) = term-to-unconst t
      fun rewrite-one (prev, const, rest) =
            let val (t', []) = term-from-unconst ucterm (prev @ [escape-const const]
@ rest)
            in improved t' end
      val consts-to-rewrite = focus-list consts |> map rewrite-one
      val consts' = map2 (fn rewr => fn const => if rewr then escape-const const
else const) consts-to-rewrite consts
      val (t', []) = term-from-unconst ucterm consts'
  in t' end

(* Do rewrite passes until bad strings are gone or no more rewrites are possible *)
fun desugar ctxt (t0: term) (bads: string list): term =
   let fun count t = count-matches (Symbol.explode (term-to-string ctxt true t))
(map Symbol.explode bads)
      val - = if null bads then error Nothing to desugar else ()
      fun rewrite t = let
        val counts0 = count t
        fun improved t' = exists (<) (count t' ~~ counts0)
        val t' = rewrite-pass ctxt t improved escape-const
        in if forall (fn c => c = 0) (count t') (* bad strings gone *)
           then t'
           else if t = t' (* no more rewrites *)
             then let
               val bads' = filter (fn (c, -) => c > 0) (counts0 ~~ bads) |> map snd
               val - = warning (Sorry, failed to desugar  ^ commas-quote bads')
               in t end
             else rewrite t'
         end
  in rewrite t0 end

fun span - [] = ([],[])
  | span p (a::s) =
      if p a then let val (y, n) = span p s in (a::y, n) end else ([], a::s)

fun check-desugar s = let
  fun replace [] = []
    | replace xs =
      if take (String.size desugar-random-tag) xs = String.explode desugar-random-tag
          then case span is-ident-char xs of
                    (v, xs) => String.explode (unescape-const (String.implode v)) @
replace xs
          else hd xs :: replace (tl xs)
  val desugar-string = String.implode o replace o String.explode
  in if not (String.isSubstring desugar-random-tag s) then s
       else desugar-string s end
```

fun desugar-term ctxt t s =
desugar ctxt t s |> term-to-string ctxt false |> check-desugar

fun desugar-thm ctxt thm s = desugar-term ctxt (Thm.prop-of thm) s

fun desugar-goal ctxt goal n s = let
val subgoals = goal |> Thm.prems-of
val subgoals = if n = 0 then subgoals else
if n < 1 orelse n > length subgoals then
(∗ trigger error ∗) [Logic.get-goal (Thm.term-of (Thm.cprop-of goal)) n]
else [nth subgoals (n − 1)]
val results = map (fn t => (NONE, desugar-term ctxt t s)
handle ex as TERM - => (SOME ex, term-to-string ctxt false t))
subgoals
in if null results
then error No subgoals to desugar
else if forall (Option.isSome o fst) results
then raise the (fst (hd results))
else map snd results
end

end
⟩

ML ⟨
Outer-Syntax.command @{command-keyword desugar-term}
term str str2... −> desugar str in term
(Parse.term −− Scan.repeat1 Parse.string >> (fn (t, s) =>
Toplevel.keep (fn state => let val ctxt = Toplevel.context-of state in
Insulin.desugar-term ctxt (Syntax.read-term ctxt t) s
|> writeln end)))
⟩

ML ⟨
Outer-Syntax.command @{command-keyword desugar-thm}
thm str str2... −> desugar str in thm
(Parse.thm −− Scan.repeat1 Parse.string >> (fn (t, s) =>
Toplevel.keep (fn state => let val ctxt = Toplevel.context-of state in
Insulin.desugar-thm ctxt (Attrib.eval-thms ctxt [t] |> hd) s |> writeln end)))
⟩

ML ⟨
fun print-subgoals (x::xs) n = (writeln (Int.toString n ˆ . ˆ x); print-subgoals xs (n+1))
| print-subgoals [] - = ();
Outer-Syntax.command @{command-keyword desugar-goal}

```
  goal-num str str2... −> desugar str in goal
  (Scan.option Parse.int −− Scan.repeat1 Parse.string >> (fn (n, s) =>
    Toplevel.keep (fn state => let val ctxt = Toplevel.context-of state in
      Insulin.desugar-goal ctxt (Toplevel.proof-of state |> Proof.raw-goal |> #goal)
(Option.getOpt (n, 0)) s
      |> (fn xs => case xs of
            [x] => writeln x
            | - => print-subgoals xs 1) end)))
⟩
```

end

theory *ShowTypes* **imports**
  Main
keywords *term-show-types thm-show-types goal-show-types* :: *diag*
begin

```
ML ⟨
structure Show-Types = struct

fun pretty-markup-to-string no-markup =
  Pretty.string-of
  #> YXML.parse-body
  #> (if no-markup then XML.content-of else YXML.string-of-body)

fun term-show-types no-markup ctxt term =
  let val keywords = Thy-Header.get-keywords′ ctxt
      val ctxt′ = ctxt
      |> Config.put show-markup false
      |> Config.put Printer.show-type-emphasis false

      (∗ FIXME: the sledgehammer code also sets these,
       ∗        but do we always want to force them on the user? ∗)
      (∗
      |> Config.put show-types false
      |> Config.put show-sorts false
      |> Config.put show-consts false
      ∗)
      |> Variable.auto-fixes term
  in
    singleton (Syntax.uncheck-terms ctxt′) term
    |> Sledgehammer-Isar-Annotate.annotate-types-in-term ctxt′
    |> Syntax.unparse-term ctxt′
    |> pretty-markup-to-string no-markup
  end
```

```
fun goal-show-types no-markup ctxt goal n = let
  val subgoals = goal |> Thm.prems-of
  val subgoals = if n = 0 then subgoals else
                 if n < 1 orelse n > length subgoals then
                      (* trigger error *) [Logic.get-goal (Thm.term-of (Thm.cprop-of
goal)) n]
                 else [nth subgoals (n − 1)]
  val results = map (fn t => (NONE, term-show-types no-markup ctxt t)
                              handle ex as TERM - => (SOME ex, term-show-types
no-markup ctxt t))
                   subgoals
  in if null results
        then error No subgoals to show
     else if forall (Option.isSome o fst) results
        then raise the (fst (hd results))
     else map snd results
  end

end;

Outer-Syntax.command @{command-keyword term-show-types}
  term-show-types TERM −> show TERM with type annotations
  (Parse.term >> (fn t =>
    Toplevel.keep (fn state =>
      let val ctxt = Toplevel.context-of state in
        Show-Types.term-show-types false ctxt (Syntax.read-term ctxt t)
        |> writeln end)));

Outer-Syntax.command @{command-keyword thm-show-types}
  thm-show-types THM1 THM2 ... −> show theorems with type annotations
  (Parse.thms1 >> (fn ts =>
    Toplevel.keep (fn state =>
      let val ctxt = Toplevel.context-of state in
        Attrib.eval-thms ctxt ts
      |> app (Thm.prop-of #> Show-Types.term-show-types false ctxt #> writeln)
end)));

let
  fun print-subgoals (x::xs) n = (writeln (Int.toString n ^ .  ^ x); print-subgoals xs
(n+1))
    | print-subgoals [] - = ();
in
Outer-Syntax.command @{command-keyword goal-show-types}
  goal-show-types [N] −> show subgoals (or Nth goal) with type annotations
  (Scan.option Parse.int >> (fn n =>
    Toplevel.keep (fn state =>
      let val ctxt = Toplevel.context-of state
          val goal = Toplevel.proof-of state |> Proof.raw-goal |> #goal
      in Show-Types.goal-show-types false ctxt goal (Option.getOpt (n, 0))
```

```
      |> (fn xs => case xs of
                 [x] => writeln x
               | _ => print_subgoals xs 1) end)))
end;
›
```

end

theory *AutoLevity-Base*
imports *Main Apply-Trace*
keywords *levity-tag* :: *thy-decl*
begin

ML ‹
```
fun is-simp (-: Proof.context) (-: thm) = true
›
```

ML ‹
```
val is-simp-installed = is-some (
 try (ML-Context.eval ML-Compiler.flags @{here})
  (ML-Lex.read-text (val is-simp = Raw-Simplifier.is-simp, @{here} )));
›
```

ML‹
```
(* Describing a ordering on Position.T. Optionally we compare absolute document
position, or
   just line numbers. Somewhat complicated by the fact that jEdit positions don't
have line or
   file identifiers. *)

fun pos-ord use-offset (pos1, pos2) =
  let
    fun get-offset pos = if use-offset then Position.offset-of pos else SOME 0;

    fun get-props pos =
      (SOME (Position.file-of pos |> the,
            (Position.line-of pos |> the,
             get-offset pos |> the)), NONE)
      handle Option => (NONE, Position.parse-id pos)

    val props1 = get-props pos1;
    val props2 = get-props pos2;

  in prod-ord
      (option-ord (prod-ord string-ord (prod-ord int-ord int-ord)))
      (option-ord (int-ord))
      (props1, props2) end
```

```
structure Postab = Table(type key = Position.T val ord = (pos-ord false));
structure Postab-strict = Table(type key = Position.T val ord = (pos-ord true));


signature AUTOLEVITY-BASE =
sig
type extras = {levity-tag : string option, subgoals : int}



val get-transactions : unit −> ((string ∗ extras) Postab-strict.table ∗ string list
Postab-strict.table) Symtab.table;

val get-applys : unit −> ((string ∗ string list) list) Postab-strict.table Symtab.table;

val add-attribute-test: string −> (Proof.context −> thm −> bool) −> theory −>
theory;

val attribs-of: Proof.context −> thm −> string list;

val used-facts: Proof.context option −> thm −> (string ∗ thm) list;
val used-facts-attribs: Proof.context −> thm −> (string ∗ string list) list;

(∗
  Returns the proof body form of the prop proved by a theorem.

  Unfortunately, proof bodies don't contain terms in the same form as what you'd
get
  from things like ‘Thm.full-prop-of‘: the proof body terms have sort constraints
  pulled out as separate assumptions, rather than as annotations on the types of
  terms.

  It's easier for our dependency−tracking purposes to treat this transformed
  term as the 'canonical' form of a theorem, since it's always available as the
  top−level prop of a theorem's proof body.
∗)
val proof-body-prop-of: thm −> term;

(∗
  Get every (named) term that was proved in the proof body of the given thm.

  The returned terms are in proof body form.
∗)
val used-named-props-of: thm −> (string ∗ term) list;

(∗
  Distinguish whether the thm name foo-3 refers to foo(3) or foo-3 by comparing
  against the given term. Assumes the term is in proof body form.
```

```
  The provided context should match the context used to extract the (name, prop)
pair
  (that is, it should match the context used to extract the thm passed into
  ‘proof-body-prop-of‘ or ‘used-named-props-of‘).

  Returns SOME (foo, SOME 3) if the answer is 'it refers to foo(3)'.
  Returns SOME (foo-3, NONE) if the answer is 'it refers to foo-3'.
  Returns NONE if the answer is 'it doesn't seem to refer to anything.'
*)
val disambiguate-indices: Proof.context -> string * term -> (string * int option)
option;

(* Install toplevel hook for tracking command positions. *)

val setup-command-hook: {trace-apply : bool} -> theory -> theory;

(* Used to trace the dependencies of all apply statements.
   They are set up by setup-command-hook if the appropriate hooks in the Proof
   module exist. *)

val pre-apply-hook: Proof.context -> Method.text -> thm -> thm;
val post-apply-hook: Proof.context -> Method.text -> thm -> thm -> thm;


end;



structure AutoLevity-Base : AUTOLEVITY-BASE =
struct

val applys = Synchronized.var applys
  (Symtab.empty : (((string * string list) list) Postab-strict.table) Symtab.table)

fun get-applys () = Synchronized.value applys;

type extras = {levity-tag : string option, subgoals : int}


val transactions = Synchronized.var hook
 (Symtab.empty : ((string * extras) Postab-strict.table * ((string list) Postab-strict.table))
Symtab.table);

fun get-transactions () =
  Synchronized.value transactions;
```

```
structure Data = Theory-Data
  (
    type T = (bool *
        string option *
       (Proof.context -> thm -> bool) Symtab.table); (* command-hook * levity
tag * attribute tests *)
    val empty = (false, NONE, Symtab.empty);
    val extend = I;
     fun merge (((b1, -, tab), (b2, -, tab')) : T * T) = (b1 orelse b2, NONE,
Symtab.merge (fn - => true) (tab, tab'));
  );

val set-command-hook-flag = Data.map (@{apply 3(1)} (fn - => true));
val get-command-hook-flag = #1 o Data.get

fun set-levity-tag tag = Data.map (@{apply 3(2)} (fn - => tag));
val get-levity-tag = #2 o Data.get

fun update-attrib-tab f = Data.map (@{apply 3(3)} f);


fun add-attribute-test nm f =
let
  val f' = (fn ctxt => fn thm => the-default false (try (f ctxt) thm))
in update-attrib-tab (Symtab.update-new (nm,f')) end;

val get-attribute-tests = Symtab.dest o #3 o Data.get;

(* Internal fact names get the naming scheme foo-3 to indicate the third
   member of the multi−thm foo. We need to do some work to guess if
   such a fact refers to an indexed multi−thm or a real fact named foo-3 *)

fun base-and-index nm =
let
  val exploded = space-explode - nm;
  val base =
    (exploded, (length exploded) − 1)
      |> try (List.take #> space-implode -)
      |> Option.mapPartial (Option.filter (fn nm => nm <> ))
  val idx = exploded |> try (List.last #> Int.fromString) |> Option.join;
in
  case (base, idx) of
    (SOME base, SOME idx) => SOME (base, idx)
  | - => NONE
end

fun maybe-nth idx xs = idx |> try (curry List.nth xs)

fun fact-from-derivation ctxt prop xnm =
```

```
let
  val facts = Proof-Context.facts-of ctxt;
  (* TODO: Check that exported local fact is equivalent to external one *)
  fun check-prop thm = Thm.full-prop-of thm = prop

  fun entry (name, idx) =
    (name, Position.none)
      |> try (Facts.retrieve (Context.Proof ctxt) facts)
      |> Option.mapPartial (#thms #> maybe-nth (idx - 1))
      |> Option.mapPartial (Option.filter check-prop)
      |> Option.map (pair name)

  val idx-result = (base-and-index xnm) |> Option.mapPartial entry
  val non-idx-result = (xnm, 1) |> entry

  val - =
    if is-some idx-result andalso is-some non-idx-result
    then warning (
       Levity: found two possible results for name  ^ quote xnm ^  with the same
prop:\n ^
      (@{make-string} (the idx-result)) ^ ,\nand\n ^
      (@{make-string} (the non-idx-result)) ^ .\nUsing the first one.)
    else ()
in
  merge-options (idx-result, non-idx-result)
end


(* Local facts (from locales) aren't marked in proof bodies, we only
   see their external variants. We guess the local name from the external one
   (i.e. Theory-Name.Locale-Name.foo -> foo)

   This is needed to perform localized attribute tests (e.g.. is this locale assumption
marked as simp?) *)

(* TODO: extend-locale breaks this naming scheme by adding the chunk qualifier.
This can
   probably just be handled as a special case *)

fun most-local-fact-of ctxt xnm prop =
let
  val local-name = xnm |> try (Long-Name.explode #> tl #> tl #> Long-Name.implode)
   val local-result = local-name |> Option.mapPartial (fact-from-derivation ctxt
prop)
  fun global-result () = fact-from-derivation ctxt prop xnm
in
  if is-some local-result then local-result else global-result ()
end
```

```
fun thms-of (PBody {thms,...}) = thms

(∗ We recursively descend into the proof body to find dependent facts.
   We skip over empty derivations or facts that we fail to find, but recurse
  into their dependents. This ensures that an attempt to re−build the proof dependencies
   graph will result in a connected graph. ∗)

fun proof-body-deps
  (filter-name: string −> bool)
  (get-fact: string −> term −> (string ∗ thm) option)
  (thm-ident, thm-node)
  (tab: (string ∗ thm) option Inttab.table) =
let
  val name = Proofterm.thm-node-name thm-node
  val body = Proofterm.thm-node-body thm-node
  val prop = Proofterm.thm-node-prop thm-node
  val result = if filter-name name then NONE else get-fact name prop
  val is-new-result = not (Inttab.defined tab thm-ident)
  val insert = if is-new-result then Inttab.update (thm-ident, result) else I
  val descend =
    if is-new-result andalso is-none result
    then fold (proof-body-deps filter-name get-fact) (thms-of (Future.join body))
    else I
in
  tab |> insert |> descend
end

fun used-facts opt-ctxt thm =
let
  val nm = Thm.get-name-hint thm;
  val get-fact =
    case opt-ctxt of
      SOME ctxt => most-local-fact-of ctxt
    | NONE => fn name => fn - => (SOME (name, Drule.dummy-thm));
  val body = thms-of (Thm.proof-body-of thm);
  fun filter-name nm′ = nm′ =  orelse nm′ = nm;
in
  fold (proof-body-deps filter-name get-fact) body Inttab.empty
    |> Inttab.dest |> map-filter snd
end

fun attribs-of ctxt =
let
  val tests = get-attribute-tests (Proof-Context.theory-of ctxt)
  |> map (apsnd (fn test => test ctxt));

in (fn t => map-filter (fn (testnm, test) => if test t then SOME testnm else
NONE) tests) end;
```

```
fun used-facts-attribs ctxt thm =
let
  val fact-nms = used-facts (SOME ctxt) thm;

  val attribs-of = attribs-of ctxt;

in map (apsnd attribs-of) fact-nms end

local
  fun app3 f g h x = (f x, g x, h x);

  datatype ('a, 'b) Either =
      Left of 'a
    | Right of 'b;

  local
    fun partition-map-foldr f (x, (ls, rs)) =
      case f x of
        Left l => (l :: ls, rs)
      | Right r => (ls, r :: rs);
  in
    fun partition-map f = List.foldr (partition-map-foldr f) ([], []);
  end

  (*
    Extracts the bits we care about from a thm-node: the name, the prop,
    and (the next steps of) the proof.
  *)
  val thm-node-dest =
    app3
      Proofterm.thm-node-name
      Proofterm.thm-node-prop
      (Proofterm.thm-node-body #> Future.join);

  (*
    Partitioning function for thm-node data. We want to insert any named props,
   then recursively find the named props used by any unnamed intermediate/anonymous
props.
  *)
  fun insert-or-descend (name, prop, proof) =
    if name =  then Right proof else Left (name, prop);

  (*
    Extracts the next layer of proof data from a proof step.
  *)
  val next-level = thms-of #> List.map (snd #> thm-node-dest);

  (*
    Secretly used as a set, using `()` as the values.
```

```
  *)
  structure NamePropTab = Table(
    type key = string * term;
    val ord = prod-ord fast-string-ord Term-Ord.fast-term-ord);

  val insert-all = List.foldr (fn (k, tab) => NamePropTab.update (k, ()) tab)

  (*
      Proofterm.fold-body-thms unconditionally recursively descends into the proof body,
    so instead of only getting the topmost named props we'd get -all- of them. Here
    we do a more controlled recursion.
  *)
  fun used-props-foldr (proof, named-props) =
    let
      val (to-insert, child-proofs) =
        proof |> next-level |> partition-map insert-or-descend;
      val thms = insert-all named-props to-insert;
    in
      List.foldr used-props-foldr thms child-proofs
    end;

  (*
    Extracts the outermost proof step of a thm (which is just the proof of the prop of the thm).
  *)
  val initial-proof =
    Thm.proof-body-of
      #> thms-of
      #> List.hd
      #> snd
      #> Proofterm.thm-node-body
      #> Future.join;

in
  fun used-named-props-of thm =
    let val used-props = used-props-foldr (initial-proof thm, NamePropTab.empty);
    in used-props |> NamePropTab.keys
    end;
end

val proof-body-prop-of =
  Thm.proof-body-of
    #> thms-of
    #> List.hd
    #> snd
    #> Proofterm.thm-node-prop

local
```

```
  fun thm-matches prop thm = proof-body-prop-of thm = prop

  fun entry ctxt prop (name, idx) =
    name
      |> try (Proof-Context.get-thms ctxt)
      |> Option.mapPartial (maybe-nth (idx − 1))
      |> Option.mapPartial (Option.filter (thm-matches prop))
      |> Option.map (K (name, SOME idx))

  fun warn-if-ambiguous
      name
      (idx-result: (string ∗ int option) option)
      (non-idx-result: (string ∗ int option) option) =
    if is-some idx-result andalso is-some non-idx-result
    then warning (
       Levity: found two possible results for name  ^ quote name ^  with the same
prop:\n ^
      (@{make-string} (the idx-result)) ^ ,\nand\n ^
      (@{make-string} (the non-idx-result)) ^ .\nUsing the first one.)
    else ()

in
  fun disambiguate-indices ctxt (name, prop) =
    let
      val entry = entry ctxt prop
      val idx-result = (base-and-index name) |> Option.mapPartial entry
      val non-idx-result = (name, 1) |> entry |> Option.map (apsnd (K NONE))
      val - = warn-if-ambiguous name idx-result non-idx-result
    in
      merge-options (idx-result, non-idx-result)
    end
end

(∗ We identify apply applications by the document position of their corresponding
method.
   We can only get a document position out of real methods, so internal methods
   (i.e. Method.Basic) won't have a position.∗)

fun get-pos-of-text' (Method.Source src) = SOME (snd (Token.name-of-src src))
  | get-pos-of-text' (Method.Combinator (-, -, texts)) = get-first get-pos-of-text'
texts
  | get-pos-of-text' - = NONE

(∗ We only want to apply our hooks in batch mode, so we test if our position has a
line number
   (in jEdit it will only have an id number) ∗)

fun get-pos-of-text text = case get-pos-of-text' text of
  SOME pos => if is-some (Position.line-of pos) then SOME pos else NONE
```

```
  | NONE => NONE

(* Clear the theorem dependencies using the apply-trace oracle, then
   pick up the new ones after the apply step is finished. *)

fun pre-apply-hook ctxt text thm =
  case get-pos-of-text text of NONE => thm
  | SOME - =>
      if Apply-Trace.can-clear (Proof-Context.theory-of ctxt)
      then Apply-Trace.clear-deps thm
      else thm;


val post-apply-hook = (fn ctxt => fn text => fn pre-thm => fn post-thm =>
  case get-pos-of-text text of NONE => post-thm
  | SOME pos => if Apply-Trace.can-clear (Proof-Context.theory-of ctxt) then
    (let
      val thy-nm = Context.theory-name (Thm.theory-of-thm post-thm);

      val used-facts = the-default [] (try (used-facts-attribs ctxt) post-thm);
      val - =
        Synchronized.change applys
         (Symtab.map-default
           (thy-nm, Postab-strict.empty) (Postab-strict.update (pos, used-facts)))

     (* We want to keep our old theorem dependencies around, so we put them back
into
         the goal thm when we are done *)

      val post-thm' = post-thm
        |> Apply-Trace.join-deps pre-thm

    in post-thm' end)
    else post-thm)

(* The Proof hooks need to be patched in to track apply dependencies, but the rest
of levity
   can work without them. Here we graciously fail if the hook interface is missing
*)

fun setup-pre-apply-hook () =
 try (ML-Context.eval ML-Compiler.flags @{here})
 (ML-Lex.read-text (Proof.set-pre-apply-hook AutoLevity-Base.pre-apply-hook, @{here}));

fun setup-post-apply-hook () =
 try (ML-Context.eval ML-Compiler.flags @{here})
   (ML-Lex.read-text (Proof.set-post-apply-hook AutoLevity-Base.post-apply-hook,
@{here}));
```

```
(* This command is treated specially by AutoLevity-Theory-Report. The command
executed directly
   after this one will be tagged with the given tag *)

val - =
  Outer-Syntax.command @{command-keyword levity-tag} tag for levity
    (Parse.string >> (fn str =>
      Toplevel.local-theory NONE NONE
        (Local-Theory.raw-theory (set-levity-tag (SOME str)))))

fun get-subgoals' state =
let
  val proof-state = Toplevel.proof-of state;
  val {goal, ...} = Proof.raw-goal proof-state;
in Thm.nprems-of goal end

fun get-subgoals state = the-default ~1 (try get-subgoals' state);

fun setup-toplevel-command-hook () =
Toplevel.add-hook (fn transition => fn start-state => fn end-state =>
  let val name = Toplevel.name-of transition
      val pos = Toplevel.pos-of transition;
      val thy = Toplevel.theory-of start-state;
      val thynm = Context.theory-name thy;
      val end-thy = Toplevel.theory-of end-state;
 in
  if name = clear-deps orelse name = dummy-apply orelse Position.line-of pos =
NONE then () else
  (let

    val levity-input = if name = levity-tag then get-levity-tag end-thy else NONE;

    val subgoals = get-subgoals start-state;

    val entry = {levity-tag = levity-input, subgoals = subgoals}

    val - =
      Synchronized.change transactions
          (Symtab.map-default (thynm, (Postab-strict.empty, Postab-strict.empty))
              (apfst (Postab-strict.update (pos, (name, entry)))))
  in () end) handle e => if Exn.is-interrupt e then Exn.reraise e else
    Synchronized.change transactions
          (Symtab.map-default (thynm, (Postab-strict.empty, Postab-strict.empty))
              (apsnd (Postab-strict.map-default (pos, []) (cons (@{make-string}
e)))))
  end)

fun setup-attrib-tests theory = if not (is-simp-installed) then
error Missing interface into Raw-Simplifier. Can't trace apply statements with un-
```

```
patched isabelle.
else
let
  fun is-first-cong ctxt thm =
    let
      val simpset = Raw-Simplifier.internal-ss (Raw-Simplifier.simpset-of ctxt);
      val (congs, -) = #congs simpset;
      val cong-thm = #mk-cong (#mk-rews simpset) ctxt thm;
    in
      case (find-first (fn (-, thm') => Thm.eq-thm-prop (cong-thm, thm')) congs)
of
        SOME (nm, -) =>
          Thm.eq-thm-prop (find-first (fn (nm', -) => nm' = nm) congs |> the |>
snd, cong-thm)
      | NONE => false
    end

  fun is-classical proj ctxt thm =
    let
      val intros = proj (Classical.claset-of ctxt |> Classical.rep-cs);
      val results = Item-Net.retrieve intros (Thm.full-prop-of thm);
    in exists (fn (thm', -, -) => Thm.eq-thm-prop (thm',thm)) results end
in
 theory
|> add-attribute-test simp is-simp
|> add-attribute-test cong is-first-cong
|> add-attribute-test intro (is-classical #unsafeIs)
|> add-attribute-test intro! (is-classical #safeIs)
|> add-attribute-test elim (is-classical #unsafeEs)
|> add-attribute-test elim! (is-classical #safeEs)
|> add-attribute-test dest (fn ctxt => fn thm => is-classical #unsafeEs ctxt
(Tactic.make-elim thm))
|> add-attribute-test dest! (fn ctxt => fn thm => is-classical #safeEs ctxt (Tactic.make-elim
thm))
end


fun setup-command-hook {trace-apply, ...} theory =
if get-command-hook-flag theory then theory else
let
  val - = if trace-apply then
    (the (setup-pre-apply-hook ());
     the (setup-post-apply-hook ()))
        handle Option => error Missing interface into Proof module. Can't trace
apply statements with unpatched isabelle
    else ()

  val - = setup-toplevel-command-hook ();
```

```
  val theory' = theory
    |> trace-apply ? setup-attrib-tests
    |> set-command-hook-flag

in theory' end;


end
⟩
```

end

theory *AutoLevity-Theory-Report*
imports *AutoLevity-Base*
begin

```
ML ⟨
(* An antiquotation for creating json−like serializers for
   simple records. Serializers for primitive types are automatically used,
   while serializers for complex types are given as parameters. *)
val JSON-string-encode: string −> string =
  String.translate (
      fn #\\ => \\\\
       | #\n => \\n
       | x => if Char.isPrint x then String.str x else
               \\u ^ align-right 0 4 (Int.fmt StringCvt.HEX (Char.ord x)))
  #> quote;

fun JSON-int-encode (i: int): string =
  if i < 0 then − ^ Int.toString (~i) else Int.toString i

val - = Theory.setup(
ML-Antiquotation.inline @{binding string-record}
  (Scan.lift
    (Parse.name −−|
      Parse.$$$ = −−
      Parse.position Parse.string) >>
    (fn (name,(source,pos)) =>

    let

      val entries =
      let
        val chars = String.explode source
          |> filter-out (fn #\n => true | - => false)

        val trim =
        String.explode
```

```
    #> chop-prefix (fn #  => true | - => false)
    #> snd
    #> chop-suffix (fn #  => true | - => false)
    #> fst
    #> String.implode

    val str = String.implode chars
      |> String.fields (fn #, => true | #: => true | - => false)
      |> map trim


    fun pairify [] = []
      | pairify (a::b::l) = ((a,b) :: pairify l)
      | pairify -  = error (Record syntax error ^ Position.here pos)

  in
    pairify str
  end

  val typedecl =
  type  ^ name ^ = {
  ^ (map (fn (nm,typ) => nm ^ : ^ typ) entries |> String.concatWith ,)
  ^ };

  val base-typs = [string,int,bool, string list]


  val encodes = map snd entries |> distinct (op =)
    |> filter-out (member (op =) base-typs)

  val sanitize = String.explode
  #> map (fn #  => #-
          | #. => #-
          | #* => #P
          | #( => #B
          | #) => #R
          | x => x)
  #> String.implode

  fun mk-encode typ =
  if typ = string
  then JSON-string-encode
  else if typ = int
  then JSON-int-encode
  else if typ = bool
  then Bool.toString
  else if typ = string list
 then (fn xs => (enclose \[\ \]\ (String.concatWith \, \ (map JSON-string-encode
xs))))
```

else (*sanitize typ*) ˆ -*encode*

fun mk-elem nm - value =
(*ML-Syntax.print-string* (*JSON-string-encode nm*) ˆ ˆ \ : \ ) ˆ ˆ (ˆ *value* ˆ)

fun mk-head body =
(\ ˆ {\ ˆ *String.concatWith* \, \ (ˆ *body* ˆ) ˆ \}\)

val global-head = *if* (*null encodes*) *then* *else*
fn (ˆ (*map mk-encode encodes* |> *String.concatWith* ,) ˆ) =>

val encode-body =
fn { ˆ (*map fst entries* |> *String.concatWith* ,) ˆ } : ˆ *name* ˆ => ˆ
mk-head
(*ML-Syntax.print-list* (*fn* (*field*,*typ*) => *mk-elem field typ* (*mk-encode typ* ˆ ˆ *field*)) *entries*)

val val-expr =
val (ˆ *name* ˆ -*encode*) = (
ˆ *global-head* ˆ (ˆ *encode-body* ˆ))

val - = @{*print*} *val-expr*

in
typedecl ˆ *val-expr*
end)))
⟩

ML ⟨

@{*string-record deps* = *consts* : *string list*, *types*: *string list*}
@{*string-record lemma-deps* = *consts*: *string list*, *types*: *string list*, *lemmas*: *string list*}
@{*string-record location* = *file* : *string*, *start-line* : *int*, *end-line* : *int*}
@{*string-record levity-tag* = *tag* : *string*, *location* : *location*}
@{*string-record apply-dep* = *name* : *string*, *attribs* : *string list*}

@{*string-record proof-command* =
command-name : *string*, *location* : *location*, *subgoals* : *int*, *depth* : *int*,
apply-deps : *apply-dep list* }

@{*string-record lemma-entry* =
name : *string*, *command-name* : *string*, *levity-tag* : *levity-tag option*, *location* : *location*,

```
  proof-commands : proof-command list,
  deps : lemma-deps}

@{string-record dep-entry =
  name : string, command-name : string, levity-tag : levity-tag option, location:
location,
  deps : deps}

@{string-record theory-entry =
  name : string, file : string}

@{string-record log-entry =
  errors : string list, location : location}

fun encode-list enc x = [ ^ (String.concatWith , (map enc x)) ^ ]

fun encode-option enc (SOME x) = enc x
  | encode-option - NONE = {}

val opt-levity-tag-encode = encode-option (levity-tag-encode location-encode);

val proof-command-encode = proof-command-encode (location-encode, encode-list
apply-dep-encode);

val lemma-entry-encode = lemma-entry-encode
 (opt-levity-tag-encode, location-encode, encode-list proof-command-encode, lemma-deps-encode)

val dep-entry-encode = dep-entry-encode
 (opt-levity-tag-encode, location-encode, deps-encode)

val log-entry-encode = log-entry-encode (location-encode)

›
```

ML ‹

```
signature AUTOLEVITY-THEORY-REPORT =
sig
val get-reports-for-thy: theory ->
  string * log-entry list * theory-entry list * lemma-entry list * dep-entry list *
dep-entry list

val string-reports-of:
  string * log-entry list * theory-entry list * lemma-entry list * dep-entry list *
dep-entry list
  -> string list

end;
```

```
structure AutoLevity-Theory-Report : AUTOLEVITY-THEORY-REPORT =
struct

fun map-pos-line f pos =
let
  val line = Position.line-of pos |> the;
  val file = Position.file-of pos |> the;

  val line' = f line;

  val - = if line' < 1 then raise Option else ();

in SOME (Position.line-file-only line' file) end handle Option => NONE


(* A Position.T table based on offsets (Postab-strict) can be collapsed into a line−based
one
   with lists of entries on for each line. This function searches such a table
   for the closest entry, either backwards (LESS) or forwards (GREATER) from
   the given position. *)

(* TODO: If everything is sane then the search depth shouldn't be necessary. In
practice
    entries won't be more than one or two lines apart, but if something has gone
wrong in the
   collection phase we might end up wasting a lot of time looking for an entry that
doesn't exist. *)

fun search-by-lines depth ord-kind f h pos = if depth = 0 then NONE else
  let
     val line-change = case ord-kind of LESS => ~1 | GREATER => 1 | - =>
raise Fail Bad relation
    val idx-change = case ord-kind of GREATER => 1 | - => 0;
  in
  case f pos of
   SOME x =>
    let
      val i = find-index (fn e => h (pos, e) = ord-kind) x;
    in if i > ~1 then SOME (List.nth(x, i + idx-change)) else SOME (hd x) end

  | NONE =>
    (case (map-pos-line (fn i => i + line-change) pos) of
      SOME pos' => search-by-lines (depth − 1) ord-kind f h pos'
     | NONE => NONE)
   end

fun location-from-range (start-pos, end-pos) =
  let
    val start-file = Position.file-of start-pos |> the;
```

```
    val end-file = Position.file-of end-pos |> the;
    val - = if start-file = end-file then () else raise Option;
    val start-line = Position.line-of start-pos |> the;
    val end-line = Position.line-of end-pos |> the;
  in
  SOME ({file = start-file, start-line = start-line, end-line = end-line} : location)
end
  handle Option => NONE

(* Here we collapse our proofs (lemma foo .. done) into single entries with start/end
positions. *)

fun get-command-ranges-of keywords thy-nm =
let
  fun is-ignored nm' = nm' = <ignored>
  fun is-levity-tag nm' = nm' = levity-tag

  fun is-proof-cmd nm' = nm' = apply orelse nm' = by orelse nm' = proof

  (* All top−level transactions for the given theory *)

  val (transactions, log) =
          Symtab.lookup (AutoLevity-Base.get-transactions ()) thy-nm
          |> the-default (Postab-strict.empty, Postab-strict.empty)
          ||> Postab-strict.dest
          |>> Postab-strict.dest

  (* Line−based position table of all apply statements for the given theory *)

  val applytab =
    Symtab.lookup (AutoLevity-Base.get-applys ()) thy-nm
    |> the-default Postab-strict.empty
    |> Postab-strict.dest
    |> map (fn (pos,e) => (pos, (pos,e)))
    |> Postab.make-list
   |> Postab.map (fn - => sort (fn ((pos,-),(pos', -)) => pos-ord true (pos, pos')))


  (* A special ignored command lets us find the real end of commands which span
     multiple lines. After finding a real command, we assume the last ignored one
     was part of the syntax for that command *)

  fun find-cmd-end last-pos ((pos', (nm', ext)) :: rest) =
    if is-ignored nm' then
       find-cmd-end pos' rest
    else (last-pos, ((pos', (nm', ext)) :: rest))
    | find-cmd-end last-pos [] = (last-pos, [])

  fun change-level nm level =
```

```
    if Keyword.is-proof-open keywords nm then level + 1
    else if Keyword.is-proof-close keywords nm then level − 1
    else if Keyword.is-qed-global keywords nm then ~1
    else level


  fun make-apply-deps lemma-deps =
    map (fn (nm, atts) => {name = nm, attribs = atts} : apply-dep) lemma-deps

  (* For a given apply statement, search forward in the document for the closest
method to retrieve
     its lemma dependencies *)

  fun find-apply pos = if Postab.is-empty applytab then [] else
    search-by-lines 5 GREATER (Postab.lookup applytab) (fn (pos, (pos', -)) =>
pos-ord true (pos, pos')) pos
   |> Option.map snd |> the-default [] |> make-apply-deps

  fun find-proof-end level ((pos', (nm', ext)) :: rest) =
    let val level' = change-level nm' level in
     if level' > ~1 then
       let
         val (cmd-end, rest') = find-cmd-end pos' rest;
         val ((prf-cmds, prf-end), rest'') = find-proof-end level' rest'
      in (({command-name = nm', location = location-from-range (pos', cmd-end)
|> the,
            depth = level, apply-deps = if is-proof-cmd nm' then find-apply pos' else
[],
            subgoals = #subgoals ext} :: prf-cmds, prf-end), rest'') end
     else
       let
         val (cmd-end, rest') = find-cmd-end pos' rest;
      in (([{command-name = nm', location = location-from-range (pos', cmd-end)
|> the,
            apply-deps = if is-proof-cmd nm' then find-apply pos' else [],
            depth = level, subgoals = #subgoals ext}], cmd-end), rest') end
     end
     | find-proof-end - - = (([], Position.none), [])


  fun find-ends tab tag ((pos,(nm, ext)) :: rest) =
   let
     val (cmd-end, rest') = find-cmd-end pos rest;

     val ((prf-cmds, pos'), rest'') =
       if Keyword.is-theory-goal keywords nm
       then find-proof-end 0 rest'
       else (([],cmd-end),rest');
```

```
    val tab' = Postab.cons_list (pos, (pos, (nm, pos', tag, prf_cmds))) tab;

    val tag' =
      if is_levity_tag nm then Option.map (rpair (pos,pos')) (#levity_tag ext) else
NONE;

  in find_ends tab' tag' rest'' end
    | find_ends tab _ [] = tab

  val command_ranges = find_ends Postab.empty NONE transactions
  |> Postab.map (fn _ => sort (fn ((pos,_),(pos',_)) => pos_ord true (pos, pos')))

in (command_ranges, log) end



fun make_deps (const_deps, type_deps): deps =
  {consts = distinct (op =) const_deps, types = distinct (op =) type_deps}

fun make_lemma_deps (const_deps, type_deps, lemma_deps): lemma_deps =
  {
    consts = distinct (op =) const_deps,
    types = distinct (op =) type_deps,
    lemmas = distinct (op =) lemma_deps
  }

fun make_tag (SOME (tag, range)) = (case location_from_range range
  of SOME rng => SOME ({tag = tag, location = rng} : levity_tag)
  | NONE => NONE)
  | make_tag NONE = NONE



fun add_deps (((Defs.Const, nm), _) :: rest) =
  let val (consts, types) = add_deps rest in
    (nm :: consts, types) end
  | add_deps (((Defs.Type, nm), _) :: rest) =
  let val (consts, types) = add_deps rest in
    (consts, nm :: types) end
  | add_deps _ = ([], [])

fun get_deps ({rhs, ...} : Defs.spec) = add_deps rhs

fun typs_of_typ (Type (nm, Ts)) = nm :: (map typs_of_typ Ts |> flat)
  | typs_of_typ _ = []

fun typs_of_term t = Term.fold_types (append o typs_of_typ) t []

fun deps_of_thm thm =
```

```
let
  val consts = Term.add-const-names (Thm.prop-of thm) [];
  val types = typs-of-term (Thm.prop-of thm);
in (consts, types) end

fun file-of-thy thy =
  let
    val path = Resources.master-directory thy;
    val name = Context.theory-name thy;
    val path′ = Path.append path (Path.basic (name ˆ .thy))
  in Path.smart-implode path′ end;

fun entry-of-thy thy = ({name = Context.theory-name thy, file = file-of-thy thy}
: theory-entry)

fun used-facts thy thm =
  AutoLevity-Base.used-named-props-of thm
    |> map-filter (AutoLevity-Base.disambiguate-indices (Proof-Context.init-global
thy))
    |> List.map fst;

fun get-reports-for-thy thy =
  let
    val thy-nm = Context.theory-name thy;
    val all-facts = Global-Theory.facts-of thy;
    val fact-space = Facts.space-of all-facts;

   val (tab, log) = get-command-ranges-of (Thy-Header.get-keywords thy) thy-nm;

    val parent-facts = map Global-Theory.facts-of (Theory.parents-of thy);

    val search-backwards = search-by-lines 5 LESS (Postab.lookup tab)
      (fn (pos, (pos′, -)) => pos-ord true (pos, pos′))
      #> the

    val lemmas =  Facts.dest-static false parent-facts (Global-Theory.facts-of thy)
    |> map-filter (fn (xnm, thms) =>
       let
          val {pos, theory-name, ...} = Name-Space.the-entry fact-space xnm;
          in
            if theory-name = thy-nm then
            let
             val thms′ = map (Thm.transfer thy) thms;

            val (real-start, (cmd-name, end-pos, tag, prf-cmds)) = search-backwards
pos

             val lemma-deps =
                  if cmd-name = datatype
```

```
              then []
              else map (used-facts thy) thms' |> flat |> distinct (op =);

          val (consts, types) = map deps-of-thm thms' |> ListPair.unzip |> apply2
flat
           val deps = make-lemma-deps (consts, types, lemma-deps)

           val location = location-from-range (real-start, end-pos) |> the;

           val (lemma-entry : lemma-entry) =
             {name  = xnm, command-name = cmd-name, levity-tag = make-tag
tag,
             location = location, proof-commands = prf-cmds, deps = deps}

          in SOME (pos, lemma-entry) end
          else NONE end handle Option => NONE)
    |> Postab-strict.make-list
    |> Postab-strict.dest |> map snd |> flat

  val defs = Theory.defs-of thy;

  fun get-deps-of kind space xnms = xnms
  |> map-filter (fn xnm =>
    let
        val {pos, theory-name, ...} = Name-Space.the-entry space xnm;
        in
          if theory-name = thy-nm then
          let
            val specs = Defs.specifications-of defs (kind, xnm);

            val deps =
              map get-deps specs
             |> ListPair.unzip
             |> (apply2 flat #> make-deps);

            val (real-start, (cmd-name, end-pos, tag, -)) = search-backwards pos

            val loc = location-from-range (real-start, end-pos) |> the;

            val entry =
              ({name = xnm, command-name = cmd-name, levity-tag = make-tag
tag,
                location = loc, deps = deps} : dep-entry)

          in SOME (pos, entry) end
          else NONE end handle Option => NONE)
    |> Postab-strict.make-list
    |> Postab-strict.dest |> map snd |> flat
```

```
    val {const-space, constants, ...} = Consts.dest (Sign.consts-of thy);

    val consts = get-deps-of Defs.Const const-space (map fst constants);

    val {types, ...} = Type.rep-tsig (Sign.tsig-of thy);

    val type-space = Name-Space.space-of-table types;
    val type-names = Name-Space.fold-table (fn (xnm, -) => cons xnm) types [];

    val types = get-deps-of Defs.Type type-space type-names;

    val thy-parents = map entry-of-thy (Theory.parents-of thy);

    val logs = log |>
     map (fn (pos, errs) => {errors = errs, location = location-from-range (pos,
pos) |> the} : log-entry)

   in (thy-nm, logs, thy-parents, lemmas, consts, types) end

fun add-commas (s :: s' :: ss) = s ^ , :: (add-commas (s' :: ss))
  | add-commas [s] = [s]
  | add-commas - = []


fun string-reports-of (thy-nm, logs, thy-parents, lemmas, consts, types) =
      [{\theory-name\ :  ^ JSON-string-encode thy-nm ^ ,] @
      [\logs\ : [] @
      add-commas (map (log-entry-encode) logs) @
      [],,\theory-imports\ : [] @
      add-commas (map (theory-entry-encode) thy-parents) @
      [],,\lemmas\ : [] @
      add-commas (map (lemma-entry-encode) lemmas) @
      [],,\consts\ : [] @
      add-commas (map (dep-entry-encode) consts) @
      [],,\types\ : [] @
      add-commas (map (dep-entry-encode) types) @
      []}]
      |> map (fn s => s ^ \n)

end
›
```

end

theory *AutoLevity-Hooks*
imports
  AutoLevity-Base
  AutoLevity-Theory-Report

begin

end

theory *Locale-Abbrev*
  imports *Main*
  keywords *revert-abbrev* :: *thy-decl* **and** *locale-abbrev* :: *thy-decl*
begin

ML ‹

```
local

fun revert-abbrev (mode,name) lthy =
  let
    val the-const = (fst o dest-Const) oo Proof-Context.read-const {proper = true,
strict = false};
  in
    Local-Theory.raw-theory (Sign.revert-abbrev (fst mode) (the-const lthy name))
lthy
  end

fun name-of spec lthy = Local-Defs.abs-def (Syntax.read-term lthy spec) |> #1 |>
#1

in

val - =
  Outer-Syntax.local-theory @{command-keyword revert-abbrev}
    make an abbreviation available for output
    (Parse.syntax-mode -- Parse.const >> revert-abbrev)

val - =
  Outer-Syntax.local-theory' @{command-keyword locale-abbrev}
    constant abbreviation that provides also provides printing in locales
    (Parse.syntax-mode -- Scan.option Parse-Spec.constdecl -- Parse.prop --
Parse.for-fixes
      >> (fn (((mode, decl), spec), params) => fn restricted => fn lthy =>
          lthy
          |> Local-Theory.open-target |> snd
          |> Specification.abbreviation-cmd mode decl params spec restricted
          |> Local-Theory.close-target (* commit new abbrev. name *)
          |> revert-abbrev (mode, name-of spec lthy)));

end
```
›

end

theory *NICTATools*
imports
  Apply-Trace-Cmd
  Apply-Debug
  Find-Names

  Rule-By-Method
  Eisbach-Methods
  TSubst
  Time-Methods-Cmd
  Try-Attribute
  Trace-Schematic-Insts
  Insulin
  ShowTypes
  AutoLevity-Hooks
  Locale-Abbrev
begin

11 Detect unused meta-forall

ML ‹

```
(* Return a list of meta−forall variable names that appear
 * to be unused in the input term. *)
fun find-unused-metaall (Const (@{const-name Pure.all}, -) $ Abs (n, -, t)) =
      (if not (Term.is-dependent t) then [n] else []) @ find-unused-metaall t
  | find-unused-metaall (Abs (-, -, t)) =
      find-unused-metaall t
  | find-unused-metaall (a $ b) =
      find-unused-metaall a @ find-unused-metaall b
  | find-unused-metaall - = []

(* Given a proof state, analyse its assumptions for unused
 * meta−foralls. *)
fun detect-unused-meta-forall - (state : Proof.state) =
let
  (* Fetch all assumptions and the main goal, and analyse them. *)
  val {context = lthy, goal = goal, ...} = Proof.goal state
  val checked-terms =
      [Thm.concl-of goal] @ map Thm.term-of (Assumption.all-assms-of lthy)
  val results = List.concat (map find-unused-metaall checked-terms)

  (* Produce a message. *)
  fun message results =
    Pretty.paragraph [
```

```
    Pretty.str Unused meta−forall(s): ,
    Pretty.commas
      (map (fn b => Pretty.mark-str (Markup.bound, b)) results)
    |> Pretty.paragraph,
    Pretty.str .
  ]

  (* We use a warning instead of the standard mechanisms so that
   * we can produce a warning icon in Isabelle/jEdit. *)
  val - =
    if length results > 0 then
      warning (message results |> Pretty.string-of)
    else ()
in
  (false, (, []))
end

(* Setup the tool, stealing the auto-solve-direct option. *)
val - = Try.tool-setup (unused-meta-forall,
    (1, @{system-option auto-solve-direct}, detect-unused-meta-forall))
›
```

lemma *test-unused-meta-forall*: $\bigwedge x.\ y \vee \neg\ y$
oops

end
Library **theory** *Lib*
imports
Value-Abbreviation
Match-Abbreviation
Try-Methods
Extract-Conjunct
Eval-Bool
NICTATools
HOL−Library.Prefix-Order
HOL−Word.Word
begin

abbreviation (*input*)
split :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a \times 'b \Rightarrow 'c$
where
split == *case-prod*

lemma *hd-map-simp*:
$b \neq [] \Longrightarrow hd\ (map\ a\ b) = a\ (hd\ b)$
by (*rule hd-map*)

lemma *tl-map-simp*:
  $tl\ (map\ a\ b) = map\ a\ (tl\ b)$
  by (*induct b*,*auto*)

lemma *Collect-eq*:
  $\{x.\ P\ x\} = \{x.\ Q\ x\} \longleftrightarrow (\forall x.\ P\ x = Q\ x)$
  by (*rule iffI*) *auto*

lemma *iff-impI*: $[\![P \Longrightarrow Q = R]\!] \Longrightarrow (P \longrightarrow Q) = (P \longrightarrow R)$ **by** *blast*

definition
  $fun\text{-}app :: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ (**infixr** $ *10*) **where**
  $f\ \$\ x \equiv f\ x$

declare *fun-app-def* [*iff*]

lemma *fun-app-cong*[*fundef-cong*]:
  $[\![\ f\ x = f'\ x'\ ]\!] \Longrightarrow (f\ \$\ x) = (f'\ \$\ x')$
  by *simp*

lemma *fun-app-apply-cong*[*fundef-cong*]:
  $f\ x\ y = f'\ x'\ y' \Longrightarrow (f\ \$\ x)\ y = (f'\ \$\ x')\ y'$
  by *simp*

lemma *if-apply-cong*[*fundef-cong*]:
  $[\![\ P = P';\ x = x';\ P' \Longrightarrow f\ x' = f'\ x';\ \neg\ P' \Longrightarrow g\ x' = g'\ x'\ ]\!]$
    $\Longrightarrow (\mathit{if}\ P\ \mathit{then}\ f\ \mathit{else}\ g)\ x = (\mathit{if}\ P'\ \mathit{then}\ f'\ \mathit{else}\ g')\ x'$
  by *simp*

lemma *case-prod-apply-cong*[*fundef-cong*]:
  $[\![\ f\ (fst\ p)\ (snd\ p)\ s = f'\ (fst\ p')\ (snd\ p')\ s'\ ]\!] \Longrightarrow case\text{-}prod\ f\ p\ s = case\text{-}prod\ f'$
$p'\ s'$
  by (*simp add*: *split-def*)

lemma *prod-injects*:
  $(x,y) = p \Longrightarrow x = fst\ p \wedge y = snd\ p$
  $p = (x,y) \Longrightarrow x = fst\ p \wedge y = snd\ p$
  by *auto*

definition
  $pred\text{-}conj :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow bool)$ (**infixl** *and 35*)
where
  $pred\text{-}conj\ P\ Q \equiv \lambda x.\ P\ x \wedge Q\ x$

definition

$pred\text{-}disj :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow bool)$ (**infixl** *or 30*)
where
$pred\text{-}disj\ P\ Q \equiv \lambda x.\ P\ x \lor Q\ x$

definition
$pred\text{-}neg :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow bool)$ $(not\ \text{-}\ [40]\ 40)$
where
$pred\text{-}neg\ P \equiv \lambda x.\ \neg\ P\ x$

definition $K \equiv \lambda x\ y.\ x$

definition
$zipWith :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a\ list \Rightarrow 'b\ list \Rightarrow 'c\ list$ **where**
$zipWith\ f\ xs\ ys \equiv map\ (case\text{-}prod\ f)\ (zip\ xs\ ys)$

primrec
$delete :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list$
where
$delete\ y\ []\ =\ []$
$|\ delete\ y\ (x\#xs) = (if\ y{=}x\ then\ xs\ else\ x\ \#\ delete\ y\ xs)$

definition
$swp\ f \equiv \lambda x\ y.\ f\ y\ x$

lemma $swp\text{-}apply[simp]$: $swp\ f\ y\ x = f\ x\ y$
by $(simp\ add:\ swp\text{-}def)$

primrec (*nonexhaustive*)
$theRight :: 'a + 'b \Rightarrow 'b$ **where**
$theRight\ (Inr\ x) = x$

primrec (*nonexhaustive*)
$theLeft :: 'a + 'b \Rightarrow 'a$ **where**
$theLeft\ (Inl\ x) = x$

definition
$isLeft\ x \equiv (\exists\, y.\ x = Inl\ y)$

definition
$isRight\ x \equiv (\exists\, y.\ x = Inr\ y)$

definition
$const\ x \equiv \lambda y.\ x$

primrec
$opt\text{-}rel :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a\ option \Rightarrow 'b\ option \Rightarrow bool$
where
$opt\text{-}rel\ f\ \ None\ \ \ \ y = (y = None)$
$|\ opt\text{-}rel\ f\ (Some\ x)\ y = (\exists\, y'.\ y = Some\ y' \land f\ x\ y')$

lemma *opt-rel-None-rhs*[*simp*]:
  opt-rel f x None = (*x* = *None*)
  by (*cases x*, *simp-all*)

lemma *opt-rel-Some-rhs*[*simp*]:
  opt-rel f x (*Some y*) = ($\exists x'$. *x* = *Some x'* $\wedge$ *f x' y*)
  by (*cases x*, *simp-all*)

lemma *tranclD2*:
  $(x, y) \in R^+ \Longrightarrow \exists z. (x, z) \in R^* \wedge (z, y) \in R$
  by (*erule tranclE*) *auto*

lemma *linorder-min-same1* [*simp*]:
  (*min y x* = *y*) = (*y* $\leq$ (*x*::'*a*::*linorder*))
  by (*auto simp*: *min-def linorder-not-less*)

lemma *linorder-min-same2* [*simp*]:
  (*min x y* = *y*) = (*y* $\leq$ (*x*::'*a*::*linorder*))
  by (*auto simp*: *min-def linorder-not-le*)

A combinator for pairing up well-formed relations. The divisor function splits the population in halves, with the True half greater than the False half, and the supplied relations control the order within the halves.

definition
  wf-sum :: ('*a* $\Rightarrow$ *bool*) $\Rightarrow$ ('*a* $\times$ '*a*) *set* $\Rightarrow$ ('*a* $\times$ '*a*) *set* $\Rightarrow$ ('*a* $\times$ '*a*) *set*
where
  wf-sum divisor r r' $\equiv$
    $(\{(x, y).\ \neg\ divisor\ x \wedge \neg\ divisor\ y\} \cap r')$
  $\cup\ \{(x, y).\ \neg\ divisor\ x \wedge divisor\ y\}$
  $\cup\ (\{(x, y).\ divisor\ x \wedge divisor\ y\} \cap r)$

lemma *wf-sum-wf*:
  $[\![$ *wf r*; *wf r'* $]\!] \Longrightarrow$ *wf* (*wf-sum divisor r r'*)
  apply (*simp add*: *wf-sum-def*)
  apply (*rule wf-Un*)+
      apply (*erule wf-Int2*)
     apply (*rule wf-subset*
            [**where** *r*=*measure* (λx. *If* (*divisor x*) *1 0*)])
      apply *simp*
     apply *clarsimp*
    apply *blast*
   apply (*erule wf-Int2*)
  apply *blast*
  done

abbreviation(*input*)
 option-map == *map-option*

lemmas *option-map-def* = *map-option-case*

lemma *False-implies-equals* [*simp*]:
$((False \Longrightarrow P) \Longrightarrow PROP\ Q) \equiv PROP\ Q$
apply (*rule equal-intr-rule*)
apply (*erule meta-mp*)
apply *simp*
apply *simp*
done

lemma *split-paired-Ball*:
$(\forall x \in A.\ P\ x) = (\forall x\ y.\ (x,y) \in A \longrightarrow P\ (x,y))$
by *auto*

lemma *split-paired-Bex*:
$(\exists x \in A.\ P\ x) = (\exists x\ y.\ (x,y) \in A \wedge P\ (x,y))$
by *auto*

lemma *delete-remove1*:
delete x xs = *remove1 x xs*
by (*induct xs*, *auto*)

lemma *ignore-if*:
(*y and z*) *s* $\Longrightarrow$ (*if x then y else z*) *s*
by (*clarsimp simp*: *pred-conj-def*)

lemma *zipWith-Nil2* :
zipWith f xs [] = []
unfolding *zipWith-def* **by** *simp*

lemma *isRight-right-map*:
isRight (*case-sum Inl* (*Inr o f*) *v*) = *isRight v*
by (*simp add*: *isRight-def split*: *sum.split*)

lemma *zipWith-nth*:
$[\![$ *n* < *min* (*length xs*) (*length ys*) $]\!] \Longrightarrow$ *zipWith f xs ys* ! *n* = *f* (*xs* ! *n*) (*ys* ! *n*)
unfolding *zipWith-def* **by** *simp*

lemma *length-zipWith* [*simp*]:
length (*zipWith f xs ys*) = *min* (*length xs*) (*length ys*)
unfolding *zipWith-def* **by** *simp*

lemma *first-in-uptoD*:
$a \le b \Longrightarrow (a::'a::order) \in \{a..b\}$
by *simp*

lemma *construct-singleton*:

$\llbracket\ S \neq \{\};\ \forall\, s{\in}S.\ \forall\, s'.\ s \neq s' \longrightarrow s' \notin S\ \rrbracket \Longrightarrow \exists\, x.\ S = \{x\}$
by *blast*

lemmas *insort-com* = *insort-left-comm*

lemma *bleeding-obvious*:
$(P \Longrightarrow \mathit{True}) \equiv (\mathit{Trueprop}\ \mathit{True})$
by (*rule*, *simp-all*)

lemma *Some-helper*:
$x = \mathit{Some}\ y \Longrightarrow x \neq \mathit{None}$
by *simp*

lemma *in-empty-interE*:
$\llbracket\ A \cap B = \{\};\ x \in A;\ x \in B\ \rrbracket \Longrightarrow \mathit{False}$
by *blast*

lemma *None-upd-eq*:
$g\ x = \mathit{None} \Longrightarrow g(x := \mathit{None}) = g$
by (*rule ext*) *simp*

lemma *exx* [*iff*]: $\exists\, x.\ x$ **by** *blast*
lemma *ExNot* [*iff*]: *Ex Not* **by** *blast*

lemma *cases-simp2* [*simp*]:
$((\neg\ P \longrightarrow Q) \wedge (P \longrightarrow Q)) = Q$
by *blast*

lemma *a-imp-b-imp-b*:
$((a \longrightarrow b) \longrightarrow b) = (a \vee b)$
by *blast*

lemma *length-neq*:
$\mathit{length}\ \mathit{as} \neq \mathit{length}\ \mathit{bs} \Longrightarrow \mathit{as} \neq \mathit{bs}$ **by** *auto*

lemma *take-neq-length*:
$\llbracket\ x \neq y;\ x \leq \mathit{length}\ \mathit{as};\ y \leq \mathit{length}\ \mathit{bs}\rrbracket \Longrightarrow \mathit{take}\ x\ \mathit{as} \neq \mathit{take}\ y\ \mathit{bs}$
by (*rule length-neq*, *simp*)

lemma *eq-concat-lenD*:
$\mathit{xs} = \mathit{ys}\ @\ \mathit{zs} \Longrightarrow \mathit{length}\ \mathit{xs} = \mathit{length}\ \mathit{ys} + \mathit{length}\ \mathit{zs}$
by *simp*

lemma *map-upt-reindex'*: $\mathit{map}\ f\ [a\ ..{<}\ b] = \mathit{map}\ (\lambda n.\ f\ (n + a - x))\ [x\ ..{<}\ x + b - a]$
by (*rule nth-equalityI*; *clarsimp simp*: *add.commute*)

lemma *map-upt-reindex*: $\mathit{map}\ f\ [a\ ..{<}\ b] = \mathit{map}\ (\lambda n.\ f\ (n + a))\ [0\ ..{<}\ b - a]$
by (*subst map-upt-reindex'* [**where** $x{=}0$]) *clarsimp*

lemma *notemptyI*:
  $x \in S \Longrightarrow S \neq \{\}$
  by *clarsimp*

lemma *setcomp-Max-has-prop*:
  assumes *a*: $P\ x$
  shows $P\ (\mathit{Max}\ \{(x::'a::\{\mathit{finite},\mathit{linorder}\}).\ P\ x\})$
proof −
  from *a* **have** $\mathit{Max}\ \{x.\ P\ x\} \in \{x.\ P\ x\}$
    by − (*rule Max-in*, *auto intro*: *notemptyI*)
  thus *?thesis* **by** *auto*
qed

lemma *cons-set-intro*:
  $\mathit{lst} = x\ \#\ \mathit{xs} \Longrightarrow x \in \mathit{set}\ \mathit{lst}$
  by *fastforce*

lemma *list-all2-conj-nth*:
  assumes *lall*: *list-all2 P as cs*
  and *rl*: $\bigwedge n.\ [\![P\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n);\ n < \mathit{length}\ \mathit{as}]\!] \Longrightarrow Q\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n)$
  shows *list-all2* $(\lambda a\ b.\ P\ a\ b \wedge Q\ a\ b)$ *as cs*
proof (*rule list-all2-all-nthI*)
  from *lall* **show** *length as* = *length cs* **..**
next
  fix *n*
  assume $n < \mathit{length}\ \mathit{as}$

  show $P\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n) \wedge Q\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n)$
  proof
    from *lall* **show** $P\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n)$ **by** (*rule list-all2-nthD*) *fact*
    thus $Q\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n)$ **by** (*rule rl*) *fact*
  qed
qed

lemma *list-all2-conj*:
  assumes *lall1*: *list-all2 P as cs*
  and *lall2*: *list-all2 Q as cs*
  shows *list-all2* $(\lambda a\ b.\ P\ a\ b \wedge Q\ a\ b)$ *as cs*
proof (*rule list-all2-all-nthI*)
  from *lall1* **show** *length as* = *length cs* **..**
next
  fix *n*
  assume $n < \mathit{length}\ \mathit{as}$

  show $P\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n) \wedge Q\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n)$
  proof
    from *lall1* **show** $P\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n)$ **by** (*rule list-all2-nthD*) *fact*
    from *lall2* **show** $Q\ (\mathit{as}\ !\ n)\ (\mathit{cs}\ !\ n)$ **by** (*rule list-all2-nthD*) *fact*

qed
qed

lemma *all-set-into-list-all2*:
assumes *lall*: $\forall x \in set\ ls.\ P\ x$
and $length\ ls = length\ ls'$
shows *list-all2* $(\lambda a\ b.\ P\ a)\ ls\ ls'$
proof (*rule list-all2-all-nthI*)
fix *n*
assume $n < length\ ls$
from *lall* **show** $P\ (ls\ !\ n)$
by (*rule bspec* [*OF* - *nth-mem*]) *fact*
qed *fact*

lemma *GREATEST-lessE*:
fixes $x :: 'a :: order$
assumes *gts*: $(GREATEST\ x.\ P\ x) < X$
and *px*: $P\ x$
and *gtst*: $\exists max.\ P\ max \wedge (\forall z.\ P\ z \longrightarrow (z \leq max))$
shows $x < X$
proof −
from *gtst* **obtain** *max* **where** *pm*: $P\ max$ **and** g': $\bigwedge z.\ P\ z \Longrightarrow z \leq max$
by *auto*

hence $(GREATEST\ x.\ P\ x) = max$
by (*auto intro*: *Greatest-equality*)

moreover have $x \leq max$ **using** *px* **by** (*rule* g')

ultimately show *?thesis* **using** *gts* **by** *simp*
qed

lemma *set-has-max*:
fixes $ls :: ('a :: linorder)\ list$
assumes *ls*: $ls \neq []$
shows $\exists max \in set\ ls.\ \forall z \in set\ ls.\ z \leq max$
using *ls*
proof (*induct ls*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*Cons l ls*)

show *?case*
proof (*cases* $ls = []$)
case *True*
thus *?thesis* **by** *simp*
next
case *False*
then obtain *max* **where** *mv*: $max \in set\ ls$ **and** *mm*: $\forall z \in set\ ls.\ z \leq max$

using *Cons.hyps*
    by *auto*
  show *?thesis*
  proof (*cases max* $\leq$ *l*)
    case *True*
    have $l \in$ *set* (l # *ls*) **by** *simp*
    thus *?thesis*
    proof
      from *mm* **show** $\forall z \in$ *set* (l # *ls*). $z \leq l$ **using** *True* **by** *auto*
    qed
  next
    case *False*
    from *mv* **have** *max* $\in$ *set* (l # *ls*) **by** *simp*
    thus *?thesis*
    proof
      from *mm* **show** $\forall z \in$ *set* (l # *ls*). $z \leq$ *max* **using** *False* **by** *auto*
    qed
  qed
 qed
qed

lemma *True-notin-set-replicate-conv*:
  True $\notin$ *set ls* = (*ls* = *replicate* (*length ls*) *False*)
  by (*induct ls*) *simp+*

lemma *Collect-singleton-eqI*:
  ($\bigwedge x.\ P\ x = (x = v)$) $\Longrightarrow$ $\{x.\ P\ x\} = \{v\}$
  by *auto*

lemma *exEI*:
  $[\![\ \exists y.\ P\ y;\ \bigwedge x.\ P\ x \Longrightarrow Q\ x\ ]\!] \Longrightarrow \exists z.\ Q\ z$
  by (*rule ex-forward*)

lemma *allEI*:
  assumes $\forall x.\ P\ x$
  assumes $\bigwedge x.\ P\ x \Longrightarrow Q\ x$
  shows $\forall x.\ Q\ x$
  using *assms* **by** (*rule all-forward*)

General lemmas that should be in the library

lemma *dom-ran*:
  $x \in$ *dom f* $\Longrightarrow$ *the* (*f x*) $\in$ *ran f*
  by (*simp add*: *dom-def ran-def*, *erule exE*, *simp*, *rule exI*, *simp*)

lemma *orthD1*:
  $[\![\ S \cap S' = \{\};\ x \in S]\!] \Longrightarrow x \notin S'$ **by** *auto*

lemma *orthD2*:
  $[\![\ S \cap S' = \{\};\ x \in S']\!] \Longrightarrow x \notin S$ **by** *auto*

lemma *distinct-element*:
$[\![b \cap d = \{\}; a \in b; c \in d]\!] \Longrightarrow a \neq c$
by *auto*

lemma *ball-reorder*:
$(\forall x \in A. \forall y \in B. P\ x\ y) = (\forall y \in B. \forall x \in A. \ P\ x\ y)$
by *auto*

lemma *hd-map*: $ls \neq [] \Longrightarrow hd\ (map\ f\ ls) = f\ (hd\ ls)$
by $(cases\ ls)$ *auto*

lemma *tl-map*: $tl\ (map\ f\ ls) = map\ f\ (tl\ ls)$
by $(cases\ ls)$ *auto*

lemma *not-NilE*:
$[\![xs \neq []; \bigwedge x\ xs'. xs = x \# xs' \Longrightarrow R]\!] \Longrightarrow R$
by $(cases\ xs)$ *auto*

lemma *length-SucE*:
$[\![length\ xs = Suc\ n; \bigwedge x\ xs'. xs = x \# xs' \Longrightarrow R]\!] \Longrightarrow R$
by $(cases\ xs)$ *auto*

lemma *map-upt-unfold*:
assumes *ab*: $a < b$
shows $map\ f\ [a ..< b] = f\ a \# map\ f\ [Suc\ a ..< b]$
using *assms upt-conv-Cons* **by** *auto*

lemma *tl-nat-list-simp*:
$tl\ [a..<b] = [a + 1 ..<b]$
by $(induct\ b,\ auto)$

lemma *image-Collect2*:
$case\text{-}prod\ f\ `\ \{x.\ P\ (fst\ x)\ (snd\ x)\} = \{f\ x\ y\ |x\ y.\ P\ x\ y\}$
by $(subst\ image\text{-}Collect)$ *simp*

lemma *image-id'*:
$id\ `\ Y = Y$
by *clarsimp*

lemma *image-invert*:
assumes *r*: $f \circ g = id$
and *g*: $B = g\ `\ A$
shows $A = f\ `\ B$
by $(simp\ add:\ g\ image\text{-}comp\ r)$

lemma *Collect-image-fun-cong*:
assumes *rl*: $\bigwedge a.\ P\ a \Longrightarrow f\ a = g\ a$
shows $\{f\ x \mid x.\ P\ x\} = \{g\ x \mid x.\ P\ x\}$

using *rl* **by** *force*

lemma *inj-on-take*:
shows *inj-on* (*take n*) $\{x.\ drop\ n\ x = k\}$
proof (*rule inj-onI*)
fix *x y*
assume *xv*: $x \in \{x.\ drop\ n\ x = k\}$
and *yv*: $y \in \{x.\ drop\ n\ x = k\}$
and *tk*: *take n x* = *take n y*

from *xv* **have** *take n x* @ *k* = *x*
using *append-take-drop-id mem-Collect-eq* **by** *auto*
moreover from *yv tk*
have *take n x* @ *k* = *y*
using *append-take-drop-id mem-Collect-eq* **by** *auto*
ultimately show *x* = *y* **by** *simp*
qed

lemma *foldr-upd-dom*:
dom (*foldr* ($\lambda p\ ps.\ ps\ (p \mapsto f\ p)$) *as g*) = *dom g* $\cup$ *set as*
proof (*induct as*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*Cons a as*)
show *?case*
proof (*cases* $a \in set\ as \vee a \in dom\ g$)
case *True*
hence *ain*: $a \in dom\ g \cup set\ as$ **by** *auto*
hence *dom g* $\cup$ *set* (*a* # *as*) = *dom g* $\cup$ *set as* **by** *auto*
thus *?thesis* **using** *Cons* **by** *fastforce*
next
case *False*
hence $a \notin (dom\ g \cup set\ as)$ **by** *simp*
hence *dom g* $\cup$ *set* (*a* # *as*) = *insert a* (*dom g* $\cup$ *set as*) **by** *simp*
thus *?thesis* **using** *Cons* **by** *fastforce*
qed
qed

lemma *foldr-upd-app*:
assumes *xin*: $x \in set\ as$
shows (*foldr* ($\lambda p\ ps.\ ps\ (p \mapsto f\ p)$) *as g*) *x* = *Some* (*f x*)
(**is** (*?f as g*) *x* = *Some* (*f x*))
using *xin*
proof (*induct as arbitrary*: *x*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*Cons a as*)
from *Cons.prems* **show** *?case* **by** (*subst foldr.simps*) (*auto intro*: *Cons.hyps*)
qed

lemma *foldr-upd-app-other*:
  assumes *xin*: $x \notin set\ as$
  shows $(foldr\ (\lambda p\ ps.\ ps\ (p \mapsto f\ p))\ as\ g)\ x = g\ x$
  (**is** $(?f\ as\ g)\ x = g\ x$)
  using *xin*
proof (*induct as arbitrary*: *x*)
  case *Nil* **thus** *?case* **by** *simp*
next
  case (*Cons a as*)
  from *Cons.prems* **show** *?case*
    by (*subst foldr.simps*) (*auto intro*: *Cons.hyps*)
qed

lemma *foldr-upd-app-if*:
  $foldr\ (\lambda p\ ps.\ ps(p \mapsto f\ p))\ as\ g = (\lambda x.\ if\ x \in set\ as\ then\ Some\ (f\ x)\ else\ g\ x)$
  by (*auto simp*: *foldr-upd-app foldr-upd-app-other*)

lemma *foldl-fun-upd-value*:
  $\bigwedge Y.\ foldl\ (\lambda f\ p.\ f(p := X\ p))\ Y\ e\ p = (if\ p {\in} set\ e\ then\ X\ p\ else\ Y\ p)$
  by (*induct e*) *simp-all*

lemma *foldr-fun-upd-value*:
  $\bigwedge Y.\ foldr\ (\lambda p\ f.\ f(p := X\ p))\ e\ Y\ p = (if\ p {\in} set\ e\ then\ X\ p\ else\ Y\ p)$
  by (*induct e*) *simp-all*

lemma *foldl-fun-upd-eq-foldr*:
  $!!m.\ foldl\ (\lambda f\ p.\ f(p := g\ p))\ m\ xs = foldr\ (\lambda p\ f.\ f(p := g\ p))\ xs\ m$
  by (*rule ext*) (*simp add*: *foldl-fun-upd-value foldr-fun-upd-value*)

lemma *Cons-eq-neq*:
  $[\![\ y = x;\ x\ \#\ xs \neq y\ \#\ ys\ ]\!] \Longrightarrow xs \neq ys$
  by *simp*

lemma *map-upt-append*:
  assumes *lt*: $x \leq y$
  and *lt2*: $a \leq x$
  shows $map\ f\ [a\ ..{<}\ y] = map\ f\ [a\ ..{<}\ x]\ @\ map\ f\ [x\ ..{<}\ y]$
proof (*subst map-append* [*symmetric*], *rule arg-cong* [**where** $f = map\ f$])
  from *lt* **obtain** *k* **where** *ky*: $x + k = y$
    by (*auto simp*: *le-iff-add*)

  thus $[a\ ..{<}\ y] = [a\ ..{<}\ x]\ @\ [x\ ..{<}\ y]$
    using *lt2*
    by (*auto intro*: *upt-add-eq-append*)
qed

lemma *Min-image-distrib*:
  assumes *minf*: $\bigwedge x\ y.\ [\![\ x \in A;\ y \in A\ ]\!] \Longrightarrow min\ (f\ x)\ (f\ y) = f\ (min\ x\ y)$

and *fa*: *finite A*
and *ane*: $A \neq \{\}$
shows *Min* $(f$ ‘ $A) = f$ $(Min\ A)$
proof −
have *rl*: $\bigwedge F.\ [\![\ F \subseteq A;\ F \neq \{\}\ ]\!] \Longrightarrow Min\ (f$ ‘ $F) = f\ (Min\ F)$
proof −
fix *F*
assume *fa*: $F \subseteq A$ **and** *fne*: $F \neq \{\}$
have *finite F* **by** (*rule finite-subset*) *fact+*

thus *?thesis F*
unfolding *min-def* **using** *fa fne fa*
proof (*induct rule*: *finite-subset-induct*)
case *empty*
thus *?case* **by** *simp*
next
case (*insert x F*)
thus *?case*
by (*cases* $F = \{\}$) (*auto dest*: *Min-in intro*: *minf*)
qed
qed

show *?thesis* **by** (*rule rl* [*OF order-refl*]) *fact+*
qed

lemma *min-of-mono'*:
assumes $(f\ a \leq f\ c) = (a \leq c)$
shows *min* $(f\ a)$ $(f\ c) = f$ $(min\ a\ c)$
unfolding *min-def*
by (*subst if-distrib* [**where** $f = f$, *symmetric*], *rule arg-cong* [**where** $f = f$], *rule if-cong* [*OF - refl refl*]) *fact+*

lemma *nat-diff-less*:
fixes x :: *nat*
shows $[\![\ x < y + z;\ z \leq x]\!] \Longrightarrow x - z < y$
using *less-diff-conv2* **by** *blast*

lemma *take-map-Not*:
(*take n* (*map Not xs*) = *take n xs*) = ($n = 0 \vee xs = []$)
by (*cases n*; *simp*) (*cases xs*; *simp*)

lemma *union-trans*:
assumes *SR*: $\bigwedge x\ y\ z.\ [\![\ (x,y) \in S;\ (y,z) \in R\ ]\!] \Longrightarrow (x,z) \in S\hat{}*$
shows $(R \cup S)\hat{}* = R\hat{}* \cup R\hat{}*\ O\ S\hat{}*$
apply (*rule set-eqI*)
apply *clarsimp*
apply (*rule iffI*)
apply (*erule rtrancl-induct*; *simp*)

```
   apply (erule disjE)
    apply (erule disjE)
     apply (drule (1) rtrancl-into-rtrancl)
     apply blast
    apply clarsimp
    apply (drule rtranclD [where R=S])
    apply (erule disjE)
     apply simp
    apply (erule conjE)
    apply (drule tranclD2)
    apply (elim exE conjE)
    apply (drule (1) SR)
    apply (drule (1) rtrancl-trans)
    apply blast
   apply (rule disjI2)
   apply (erule disjE)
    apply (blast intro: in-rtrancl-UnI)
   apply clarsimp
   apply (drule (1) rtrancl-into-rtrancl)
   apply (erule (1) relcompI)
  apply (erule disjE)
   apply (blast intro: in-rtrancl-UnI)
  apply clarsimp
  apply (blast intro: in-rtrancl-UnI rtrancl-trans)
  done
```

lemma *trancl-trancl*:
  $(R^+)^+ = R^+$
  by *auto*

Some rules for showing that the reflexive transitive closure of a relation/predicate doesn't add much if it was already transitively closed.

lemma *rtrancl-eq-reflc-trans*:
  assumes *trans*: *trans X*
  shows *rtrancl* $X = X \cup Id$
  by (*simp only*: *rtrancl-trancl-reflcl trancl-id*[*OF trans*])

lemma *rtrancl-id*:
  assumes *refl*: $Id \subseteq X$
  assumes *trans*: *trans X*
  shows *rtrancl* $X = X$
  using *refl rtrancl-eq-reflc-trans*[*OF trans*]
  by *blast*

lemma *rtranclp-eq-reflcp-transp*:
  assumes *trans*: *transp X*
  shows *rtranclp* $X = (\lambda x\ y.\ X\ x\ y \lor x = y)$
  by (*simp add*: *Enum.rtranclp-rtrancl-eq fun-eq-iff*
        rtrancl-eq-reflc-trans trans[*unfolded transp-trans*])

lemma *rtranclp-id*:
  shows *reflp X* $\Longrightarrow$ *transp X* $\Longrightarrow$ *rtranclp X* = *X*
  apply (*simp add*: *rtranclp-eq-reflcp-transp*)
  apply (*auto simp*: *fun-eq-iff elim*: *reflpD*)
  done

lemmas *rtranclp-id2* = *rtranclp-id*[*unfolded reflp-def transp-relcompp le-fun-def*]

lemma *if-1-0-0*:
  ((*if P then 1 else 0*) = (*0* :: (*'a* :: *zero-neq-one*))) = ($\neg$ *P*)
  by (*simp split*: *if-split*)

lemma *neq-Nil-lengthI*:
  Suc 0 $\leq$ *length xs* $\Longrightarrow$ *xs* $\neq$ []
  by (*cases xs*, *auto*)

lemmas *ex-with-length* = *Ex-list-of-length*

lemma *in-singleton*:
  S = {*x*} $\Longrightarrow$ *x* $\in$ *S*
  by *simp*

lemma *singleton-set*:
 x $\in$ *set* [*a*] $\Longrightarrow$ *x* = *a*
  by *auto*

lemma *take-drop-eqI*:
  assumes *t*: *take n xs* = *take n ys*
  assumes *d*: *drop n xs* = *drop n ys*
  shows *xs* = *ys*
proof −
  have *xs* = *take n xs* @ *drop n xs* **by** *simp*
  with *t d*
  have *xs* = *take n ys* @ *drop n ys* **by** *simp*
  moreover
  have *ys* = *take n ys* @ *drop n ys* **by** *simp*
  ultimately
  show *?thesis* **by** *simp*
qed

lemma *append-len2*:
  zs = *xs* @ *ys* $\Longrightarrow$ *length xs* = *length zs* − *length ys*
  by *auto*

lemma *if-flip*:
  (*if* $\neg$*P then T else F*) = (*if P then F else T*)
  by *simp*

lemma *not-in-domIff*: $f\ x = None = (x \notin dom\ f)$
  by *blast*

lemma *not-in-domD*:
  $x \notin dom\ f \Longrightarrow f\ x = None$
  by (*simp add*:*not-in-domIff*)

definition
  $graph\text{-}of\ f \equiv \{(x,y).\ f\ x = Some\ y\}$

lemma *graph-of-None-update*:
  $graph\text{-}of\ (f\ (p := None)) = graph\text{-}of\ f - \{p\} \times UNIV$
  by (*auto simp*: *graph-of-def split*: *if-split-asm*)

lemma *graph-of-Some-update*:
  $graph\text{-}of\ (f\ (p \mapsto v)) = (graph\text{-}of\ f - \{p\} \times UNIV) \cup \{(p,v)\}$
  by (*auto simp*: *graph-of-def split*: *if-split-asm*)

lemma *graph-of-restrict-map*:
  $graph\text{-}of\ (m\ |`\ S) \subseteq graph\text{-}of\ m$
  by (*simp add*: *graph-of-def restrict-map-def subset-iff*)

lemma *graph-ofD*:
  $(x,y) \in graph\text{-}of\ f \Longrightarrow f\ x = Some\ y$
  by (*simp add*: *graph-of-def*)

lemma *graph-ofI*:
  $m\ x = Some\ y \Longrightarrow (x,\ y) \in graph\text{-}of\ m$
  by (*simp add*: *graph-of-def*)

lemma *graph-of-empty* :
  $graph\text{-}of\ Map.empty = \{\}$
  by (*simp add*: *graph-of-def*)

lemma *graph-of-in-ranD*: $\forall y \in ran\ f.\ P\ y \Longrightarrow (x,y) \in graph\text{-}of\ f \Longrightarrow P\ y$
  by (*auto simp*: *graph-of-def ran-def*)

lemma *graph-of-SomeD*:
  $[\![\ graph\text{-}of\ f \subseteq graph\text{-}of\ g;\ f\ x = Some\ y\ ]\!] \Longrightarrow g\ x = Some\ y$
  unfolding *graph-of-def*
  by *auto*

lemma *in-set-zip-refl* :
  $(x,y) \in set\ (zip\ xs\ xs) = (y = x \wedge x \in set\ xs)$
  by (*induct xs*) *auto*

lemma *map-conv-upd*:
  $m\ v = None \Longrightarrow m\ o\ (f\ (x := v)) = (m\ o\ f)\ (x := None)$
  by (*rule ext*) (*clarsimp simp*: *o-def*)

lemma *sum-all-ex* [*simp*]:
$(\forall a.\ x \neq \mathit{Inl}\ a) = (\exists a.\ x = \mathit{Inr}\ a)$
$(\forall a.\ x \neq \mathit{Inr}\ a) = (\exists a.\ x = \mathit{Inl}\ a)$
by (*metis Inr-not-Inl sum.exhaust*)+

lemma *split-distrib*: $\mathit{case\text{-}prod}\ (\lambda a\ b.\ T\ (f\ a\ b)) = (\lambda x.\ T\ (\mathit{case\text{-}prod}\ (\lambda a\ b.\ f\ a\ b)\ x))$
by (*clarsimp simp*: *split-def*)

lemma *case-sum-triv* [*simp*]:
$(\mathit{case}\ x\ \mathit{of}\ \mathit{Inl}\ x \Rightarrow \mathit{Inl}\ x \mid \mathit{Inr}\ x \Rightarrow \mathit{Inr}\ x) = x$
by (*clarsimp split*: *sum.splits*)

lemma *set-eq-UNIV*: $(\{a.\ P\ a\} = \mathit{UNIV}) = (\forall a.\ P\ a)$
by *force*

lemma *allE2*:
$[\![\forall x\ y.\ P\ x\ y;\ P\ x\ y \Longrightarrow R]\!] \Longrightarrow R$
by *blast*

lemma *allE3*: $[\![\ \forall x\ y\ z.\ P\ x\ y\ z;\ P\ x\ y\ z \Longrightarrow R\ ]\!] \Longrightarrow R$
by *auto*

lemma *my-BallE*: $[\![\ \forall x \in A.\ P\ x;\ y \in A;\ P\ y \Longrightarrow Q\ ]\!] \Longrightarrow Q$
by (*simp add*: *Ball-def*)

lemma *unit-Inl-or-Inr* [*simp*]:
$\bigwedge a.\ (a \neq \mathit{Inl}\ ()) = (a = \mathit{Inr}\ ())$
$\bigwedge a.\ (a \neq \mathit{Inr}\ ()) = (a = \mathit{Inl}\ ())$
by (*case-tac a*; *clarsimp*)+

lemma *disjE-L*: $[\![\ a \vee b;\ a \Longrightarrow R;\ [\![\ \neg\ a;\ b\ ]\!] \Longrightarrow R\ ]\!] \Longrightarrow R$
by *blast*

lemma *disjE-R*: $[\![\ a \vee b;\ [\![\ \neg\ b;\ a\ ]\!] \Longrightarrow R;\ [\![\ b\ ]\!] \Longrightarrow R\ ]\!] \Longrightarrow R$
by *blast*

lemma *int-max-thms*:
$(a :: \mathit{int}) \leq \mathit{max}\ a\ b$
$(b :: \mathit{int}) \leq \mathit{max}\ a\ b$
by (*auto simp*: *max-def*)

lemma *sgn-negation* [*simp*]:
$\mathit{sgn}\ (-(x{::}\mathit{int})) = -\ \mathit{sgn}\ x$
by (*clarsimp simp*: *sgn-if*)

lemma *sgn-sgn-nonneg* [*simp*]:
$\mathit{sgn}\ (a :: \mathit{int}) * \mathit{sgn}\ a \neq -\mathit{1}$

by (*clarsimp simp*: *sgn-if*)

lemma *inj-inj-on*:
$inj\ f \Longrightarrow inj\text{-}on\ f\ A$
by (*metis injD inj-onI*)

lemma *ex-eqI*:
$[\![\bigwedge x.\ f\ x = g\ x]\!] \Longrightarrow (\exists\, x.\ f\ x) = (\exists\, x.\ g\ x)$
by *simp*

lemma *pre-post-ex*:
$[\![\exists\, x.\ P\ x;\ \bigwedge x.\ P\ x \Longrightarrow Q\ x]\!] \Longrightarrow \exists\, x.\ Q\ x$
by *auto*

lemma *ex-conj-increase*:
$((\exists\, x.\ P\ x) \wedge Q) = (\exists\, x.\ P\ x \wedge Q)$
$(R \wedge (\exists\, x.\ S\ x)) = (\exists\, x.\ R \wedge S\ x)$
by *simp+*

lemma *all-conj-increase*:
$((\ \forall\, x.\ P\ x) \wedge Q) = (\forall\, x.\ P\ x \wedge Q)$
$(R \wedge (\forall\, x.\ S\ x)) = (\forall\, x.\ R \wedge S\ x)$
by *simp+*

lemma *Ball-conj-increase*:
$xs \neq \{\} \Longrightarrow ((\ \forall\, x \in xs.\ P\ x) \wedge Q) = (\forall\, x \in xs.\ P\ x \wedge Q)$
$xs \neq \{\} \Longrightarrow (R \wedge (\forall\, x \in xs.\ S\ x)) = (\forall\, x \in xs.\ R \wedge S\ x)$
by *auto*

lemma *disjoint-subset*:
assumes $A' \subseteq A$ **and** $A \cap B = \{\}$
shows $A' \cap B = \{\}$
using *assms* **by** *auto*

lemma *disjoint-subset2*:
assumes $B' \subseteq B$ **and** $A \cap B = \{\}$
shows $A \cap B' = \{\}$
using *assms* **by** *auto*

lemma *UN-nth-mem*:
$i < length\ xs \Longrightarrow f\ (xs\ !\ i) \subseteq (\bigcup x{\in}set\ xs.\ f\ x)$
by (*metis UN-upper nth-mem*)

lemma *Union-equal*:
$f\ `\ A = f\ `\ B \Longrightarrow (\bigcup x \in A.\ f\ x) = (\bigcup x \in B.\ f\ x)$
by *blast*

lemma *UN-Diff-disjoint*:
  $i < \mathit{length}\ xs \Longrightarrow (A - (\bigcup x{\in}\mathit{set}\ xs.\ f\ x)) \cap f\ (xs\ !\ i) = \{\}$
  by (*metis Diff-disjoint Int-commute UN-nth-mem disjoint-subset*)

lemma *image-list-update*:
  $f\ a = f\ (xs\ !\ i)$
  $\Longrightarrow f\ `\ \mathit{set}\ (xs\ [i := a]) = f\ `\ \mathit{set}\ xs$
  by (*metis list-update-id map-update set-map*)

lemma *Union-list-update-id*:
  $f\ a = f\ (xs\ !\ i) \Longrightarrow (\bigcup x{\in}\mathit{set}\ (xs\ [i := a]).\ f\ x) = (\bigcup x{\in}\mathit{set}\ xs.\ f\ x)$
  by (*rule Union-equal*) (*erule image-list-update*)

lemma *Union-list-update-id'*:
  $[\![i < \mathit{length}\ xs;\ \bigwedge x.\ g\ (f\ x) = g\ x]\!]$
  $\Longrightarrow (\bigcup x{\in}\mathit{set}\ (xs\ [i := f\ (xs\ !\ i)]).\ g\ x) = (\bigcup x{\in}\mathit{set}\ xs.\ g\ x)$
  by (*metis Union-list-update-id*)

lemma *Union-subset*:
  $[\![\bigwedge x.\ x \in A \Longrightarrow (f\ x) \subseteq (g\ x)]\!] \Longrightarrow (\bigcup x \in A.\ f\ x) \subseteq (\bigcup x \in A.\ g\ x)$
  by (*metis UN-mono order-refl*)

lemma *UN-sub-empty*:
  $[\![\mathit{list\text{-}all}\ P\ xs;\ \bigwedge x.\ P\ x \Longrightarrow f\ x = g\ x]\!] \Longrightarrow (\bigcup x{\in}\mathit{set}\ xs.\ f\ x) - (\bigcup x{\in}\mathit{set}\ xs.\ g\ x) = \{\}$
  by (*simp add*: *Ball-set-list-all*[*symmetric*] *Union-subset*)

lemma *bij-betw-fun-updI*:
  $[\![x \notin A;\ y \notin B;\ \mathit{bij\text{-}betw}\ f\ A\ B]\!] \Longrightarrow \mathit{bij\text{-}betw}\ (f(x := y))\ (\mathit{insert}\ x\ A)\ (\mathit{insert}\ y\ B)$
  by (*clarsimp simp*: *bij-betw-def fun-upd-image inj-on-fun-updI split*: *if-split-asm*; *blast*)

definition
  $\mathit{bij\text{-}betw\text{-}map}\ f\ A\ B \equiv \mathit{bij\text{-}betw}\ f\ A\ (\mathit{Some}\ `\ B)$

lemma *bij-betw-map-fun-updI*:
  $[\![x \notin A;\ y \notin B;\ \mathit{bij\text{-}betw\text{-}map}\ f\ A\ B]\!]$
  $\Longrightarrow \mathit{bij\text{-}betw\text{-}map}\ (f(x \mapsto y))\ (\mathit{insert}\ x\ A)\ (\mathit{insert}\ y\ B)$
  unfolding *bij-betw-map-def* **by** *clarsimp* (*erule bij-betw-fun-updI*; *clarsimp*)

lemma *bij-betw-map-imp-inj-on*:
  $\mathit{bij\text{-}betw\text{-}map}\ f\ A\ B \Longrightarrow \mathit{inj\text{-}on}\ f\ A$
  by (*simp add*: *bij-betw-map-def bij-betw-imp-inj-on*)

lemma *bij-betw-empty-dom-exists*:
  $r = \{\} \Longrightarrow \exists\, t.\ \mathit{bij\text{-}betw}\ t\ \{\}\ r$

by (*clarsimp simp*: *bij-betw-def*)

lemma *bij-betw-map-empty-dom-exists*:
$r = \{\} \Longrightarrow \exists\, t.\ \textit{bij-betw-map}\ t\ \{\}\ r$
by (*clarsimp simp*: *bij-betw-map-def bij-betw-empty-dom-exists*)

lemma *funpow-add* [*simp*]:
fixes $f :: {'a} \Rightarrow {'a}$
shows $(f\ \hat{}\hat{}\ a)\ ((f\ \hat{}\hat{}\ b)\ s) = (f\ \hat{}\hat{}\ (a + b))\ s$
by (*metis comp-apply funpow-add*)

lemma *funpow-unfold*:
fixes $f :: {'a} \Rightarrow {'a}$
assumes $n > 0$
shows $f\ \hat{}\hat{}\ n = (f\ \hat{}\hat{}\ (n - 1)) \circ f$
by (*metis Suc-diff-1 assms funpow-Suc-right*)

lemma *relpow-unfold*: $n > 0 \Longrightarrow S\ \hat{}\hat{}\ n = (S\ \hat{}\hat{}\ (n - 1))\ O\ S$
by (*cases n*, *auto*)

definition
equiv-of $:: ({'s} \Rightarrow {'t}) \Rightarrow ({'s} \times {'s})\ \textit{set}$
where
equiv-of proj $\equiv \{(a, b).\ \textit{proj}\ a = \textit{proj}\ b\}$

lemma *equiv-of-is-equiv-relation* [*simp*]:
equiv UNIV (*equiv-of proj*)
by (*auto simp*: *equiv-of-def intro*!: *equivI refl-onI symI transI*)

lemma *in-equiv-of* [*simp*]:
$((a, b) \in \textit{equiv-of}\ f) \longleftrightarrow (f\ a = f\ b)$
by (*clarsimp simp*: *equiv-of-def*)

lemma *equiv-relation-to-projection*:
fixes $R :: ({'a} \times {'a})\ \textit{set}$
assumes *equiv*: *equiv UNIV R*
shows $\exists f :: {'a} \Rightarrow {'a}\ \textit{set}.\ \forall\, x\ y.\ f\ x = f\ y \longleftrightarrow (x, y) \in R$
apply (*rule exI* [*of* - $\lambda x.\ \{y.\ (x, y) \in R\}$])
apply *clarsimp*
apply (*case-tac* $(x, y) \in R$)
apply *clarsimp*
apply (*rule set-eqI*)

apply *clarsimp*
apply (*metis equivE sym-def trans-def equiv*)
apply (*clarsimp*)
apply (*metis UNIV-I equiv equivE mem-Collect-eq refl-on-def*)
done

lemma *range-constant* [*simp*]:
range (λ-. k) = $\{k\}$
by (*clarsimp simp*: *image-def*)

lemma *dom-unpack*:
dom (*map-of* (*map* (λx. ($f\ x$, $g\ x$)) *xs*)) = *set* (*map* (λx. $f\ x$) *xs*)
by (*simp add*: *dom-map-of-conv-image-fst image-image*)

lemma *fold-to-disj*:
fold (++) *ms a x* = *Some y* $\Longrightarrow$ ($\exists\, b \in$ *set ms*. $b\ x$ = *Some y*) $\vee$ $a\ x$ = *Some y*
by (*induct ms arbitrary*:*a x y*; *clarsimp*) *blast*

lemma *fold-ignore1*:
$a\ x$ = *Some y* $\Longrightarrow$ *fold* (++) *ms a x* = *Some y*
by (*induct ms arbitrary*:*a x y*; *clarsimp*)

lemma *fold-ignore2*:
fold (++) *ms a x* = *None* $\Longrightarrow$ $a\ x$ = *None*
by (*metis fold-ignore1 option.collapse*)

lemma *fold-ignore3*:
fold (++) *ms a x* = *None* $\Longrightarrow$ ($\forall\, b \in$ *set ms*. $b\ x$ = *None*)
by (*induct ms arbitrary*:*a x*; *clarsimp*) (*meson fold-ignore2 map-add-None*)

lemma *fold-ignore4*:
$b \in$ *set ms* $\Longrightarrow$ $b\ x$ = *Some y* $\Longrightarrow$ $\exists\, y$. *fold* (++) *ms a x* = *Some y*
using *fold-ignore3* **by** *fastforce*

lemma *dom-unpack2*:
dom (*fold* (++) *ms Map.empty*) = $\bigcup$(*set* (*map dom ms*))
apply (*induct ms*; *clarsimp simp*:*dom-def*)
apply (*rule equalityI*; *clarsimp*)
apply (*drule fold-to-disj*)
apply (*erule disjE*)
apply *clarsimp*
apply (*rename-tac b*)
apply (*erule-tac x*=*b* **in** *ballE*; *clarsimp*)
apply *clarsimp*
apply (*rule conjI*)
apply *clarsimp*
apply (*rule-tac x*=*y* **in** *exI*)
apply (*erule fold-ignore1*)
apply *clarsimp*

apply (*rename-tac y*)
apply (*erule-tac y=y* **in** *fold-ignore4*; *clarsimp*)
done

lemma *fold-ignore5*:*fold* (++) *ms a x* = *Some y* $\Longrightarrow$ *a x* = *Some y* $\vee$ ($\exists$ *b* $\in$ *set ms*. *b x* = *Some y*)
by (*induct ms arbitrary*:*a x y*; *clarsimp*) *blast*

lemma *dom-inter-nothing*:*dom f* $\cap$ *dom g* = {} $\Longrightarrow$ $\forall$ *x*. *f x* = *None* $\vee$ *g x* = *None*
by *auto*

lemma *fold-ignore6*:
f x = *None* $\Longrightarrow$ *fold* (++) *ms f x* = *fold* (++) *ms Map.empty x*
apply (*induct ms arbitrary*:*f x*; *clarsimp simp*:*map-add-def*)
by (*metis* (*no-types*, *lifting*) *fold-ignore1 option.collapse option.simps*(*4*))

lemma *fold-ignore7*:
m x = *m′ x* $\Longrightarrow$ *fold* (++) *ms m x* = *fold* (++) *ms m′ x*
apply (*case-tac m x*)
apply (*frule-tac ms=ms* **in** *fold-ignore6*)
apply (*cut-tac f=m′* **and** *ms=ms* **and** *x=x* **in** *fold-ignore6*)
apply *clarsimp+*
apply (*rename-tac a*)
apply (*cut-tac ms=ms* **and** *a=m* **and** *x=x* **and** *y=a* **in** *fold-ignore1*, *clarsimp*)
apply (*cut-tac ms=ms* **and** *a=m′* **and** *x=x* **and** *y=a* **in** *fold-ignore1*; *clarsimp*)
done

lemma *fold-ignore8*:
fold (++) *ms* [*x* $\mapsto$ *y*] = (*fold* (++) *ms Map.empty*)(*x* $\mapsto$ *y*)
apply (*rule ext*)
apply (*rename-tac xa*)
apply (*case-tac xa* = *x*)
apply *clarsimp*
apply (*rule fold-ignore1*)
apply *clarsimp*
apply (*subst fold-ignore6*; *clarsimp*)
done

lemma *fold-ignore9*:
⟦*fold* (++) *ms* [*x* $\mapsto$ *y*] *x′* = *Some z*; *x* = *x′*⟧ $\Longrightarrow$ *y* = *z*
by (*subst* (*asm*) *fold-ignore8*) *clarsimp*

lemma *fold-to-map-of*:
fold (++) (*map* (λ*x*. [*f x* $\mapsto$ *g x*]) *xs*) *Map.empty* = *map-of* (*map* (λ*x*. (*f x*, *g x*)) *xs*)
apply (*rule ext*)
apply (*rename-tac x*)
apply (*case-tac fold* (++) (*map* (λ*x*. [*f x* $\mapsto$ *g x*]) *xs*) *Map.empty x*)

apply *clarsimp*
apply (*drule fold-ignore3*)
apply (*clarsimp split*:*if-split-asm*)
apply (*rule sym*)
apply (*subst map-of-eq-None-iff*)
apply *clarsimp*
apply (*rename-tac xa*)
apply (*erule-tac x*=*xa* **in** *ballE*; *clarsimp*)
apply *clarsimp*
apply (*frule fold-ignore5*; *clarsimp split*:*if-split-asm*)
apply (*subst map-add-map-of-foldr*[**where** *m*=*Map.empty*, *simplified*])
apply (*induct xs arbitrary*:*f g*; *clarsimp split*:*if-split*)
apply (*rule conjI*; *clarsimp*)
apply (*drule fold-ignore9*; *clarsimp*)
apply (*cut-tac ms*=*map* ($\lambda x.$ [$f\ x \mapsto g\ x$]) *xs* **and** *f*=[$f\ a \mapsto g\ a$] **and** *x*=*f b* **in** *fold-ignore6*, *clarsimp*)
apply *auto*
done

lemma *if-n-0-0*:
$((\mathit{if}\ P\ \mathit{then}\ n\ \mathit{else}\ 0) \neq 0) = (P \wedge n \neq 0)$
by (*simp split*: *if-split*)

lemma *insert-dom*:
assumes *fx*: $f\ x = \mathit{Some}\ y$
shows $\mathit{insert}\ x\ (\mathit{dom}\ f) = \mathit{dom}\ f$
unfolding *dom-def* **using** *fx* **by** *auto*

lemma *map-comp-subset-dom*:
$\mathit{dom}\ (\mathit{prj} \circ_m f) \subseteq \mathit{dom}\ f$
unfolding *dom-def*
by (*auto simp*: *map-comp-Some-iff*)

lemmas *map-comp-subset-domD* = *subsetD* [*OF map-comp-subset-dom*]

lemma *dom-map-comp*:
$x \in \mathit{dom}\ (\mathit{prj} \circ_m f) = (\exists\, y\ z.\ f\ x = \mathit{Some}\ y \wedge \mathit{prj}\ y = \mathit{Some}\ z)$
by (*fastforce simp*: *dom-def map-comp-Some-iff*)

lemma *map-option-Some-eq2*:
$(\mathit{Some}\ y = \mathit{map\text{-}option}\ f\ x) = (\exists\, z.\ x = \mathit{Some}\ z \wedge f\ z = y)$
by (*metis map-option-eq-Some*)

lemma *map-option-eq-dom-eq*:
assumes *ome*: $\mathit{map\text{-}option}\ f \circ g = \mathit{map\text{-}option}\ f \circ g'$
shows $\mathit{dom}\ g = \mathit{dom}\ g'$
proof (*rule set-eqI*)
fix *x*
{

assume $x \in dom\ g$
hence $Some\ (f\ (the\ (g\ x))) = (map\text{-}option\ f \circ g)\ x$
by $(auto\ simp: map\text{-}option\text{-}case\ split: option.splits)$
also have $\ldots = (map\text{-}option\ f \circ g')\ x$ **by** $(simp\ add: ome)$
finally have $x \in dom\ g'$
by $(auto\ simp: map\text{-}option\text{-}case\ split: option.splits)$
} moreover
{
assume $x \in dom\ g'$
hence $Some\ (f\ (the\ (g'\ x))) = (map\text{-}option\ f \circ g')\ x$
by $(auto\ simp: map\text{-}option\text{-}case\ split: option.splits)$
also have $\ldots = (map\text{-}option\ f \circ g)\ x$ **by** $(simp\ add: ome)$
finally have $x \in dom\ g$
by $(auto\ simp: map\text{-}option\text{-}case\ split: option.splits)$
} ultimately show $(x \in dom\ g) = (x \in dom\ g')$ **by** *auto*
qed

lemma *cart-singleton-image*:
$S \times \{s\} = (\lambda v.\ (v,\ s))\ \text{`}\ S$
by *auto*

lemma *singleton-eq-o2s*:
$(\{x\} = set\text{-}option\ v) = (v = Some\ x)$
by $(cases\ v,\ auto)$

lemma *option-set-singleton-eq*:
$(set\text{-}option\ opt = \{v\}) = (opt = Some\ v)$
by $(cases\ opt,\ simp\text{-}all)$

lemmas *option-set-singleton-eqs*
= *option-set-singleton-eq*
trans[*OF eq-commute option-set-singleton-eq*]

lemma *map-option-comp2*:
$map\text{-}option\ (f\ o\ g) = map\text{-}option\ f\ o\ map\text{-}option\ g$
by $(simp\ add: option.map\text{-}comp\ fun\text{-}eq\text{-}iff)$

lemma *compD*:
$[\![f \circ g = f \circ g';\ g\ x = v\ ]\!] \Longrightarrow f\ (g'\ x) = f\ v$
by $(metis\ comp\text{-}apply)$

lemma *map-option-comp-eqE*:
assumes om: $map\text{-}option\ f \circ mp = map\text{-}option\ f \circ mp'$
and $p1$: $[\![\ mp\ x = None;\ mp'\ x = None\ ]\!] \Longrightarrow P$
and $p2$: $\bigwedge v\ v'.\ [\![\ mp\ x = Some\ v;\ mp'\ x = Some\ v';\ f\ v = f\ v'\ ]\!] \Longrightarrow P$
shows P
proof $(cases\ mp\ x)$
case *None*

```
  hence x ∉ dom mp by (simp add: domIff)
  hence mp' x = None by (simp add: map-option-eq-dom-eq [OF om] domIff)
  with None show ?thesis by (rule p1)
next
  case (Some v)
  hence x ∈ dom mp by clarsimp
   then obtain v' where Some': mp' x = Some v' by (clarsimp simp add:
map-option-eq-dom-eq [OF om])
  with Some show ?thesis
  proof (rule p2)
    show f v = f v' using Some' compD [OF om, OF Some] by simp
  qed
qed

lemma Some-the:
  x ∈ dom f ⟹ f x = Some (the (f x))
  by clarsimp

lemma map-comp-update:
  f ∘m (g(x ↦ v)) = (f ∘m g)(x := f v)
  by (rule ext, rename-tac y) (case-tac g y; simp)

lemma restrict-map-eqI:
  assumes req: A |` S = B |` S
  and     mem: x ∈ S
  shows   A x = B x
proof -
  from mem have A x = (A |` S) x by simp
  also have ... = (B |` S) x using req by simp
  also have ... = B x using mem by simp
  finally show ?thesis .
qed

lemma map-comp-eqI:
  assumes dm: dom g = dom g'
  and     fg: ⋀x. x ∈ dom g' ⟹ f (the (g' x)) = f (the (g x))
  shows  f ∘m g = f ∘m g'
  apply (rule ext)
  apply (case-tac x ∈ dom g)
   apply (frule subst [OF dm])
   apply (clarsimp split: option.splits)
   apply (frule domI [where m = g'])
   apply (drule fg)
   apply simp
  apply (frule subst [OF dm])
  apply clarsimp
  apply (drule not-sym)
  apply (clarsimp simp: map-comp-Some-iff)
  done
```

definition
modify-map m p f $\equiv$ *m* (*p* := *map-option f* (*m p*))

lemma *modify-map-id*:
modify-map m p id = *m*
by (*auto simp add*: *modify-map-def map-option-case split*: *option.splits*)

lemma *modify-map-addr-com*:
assumes *com*: $x \neq y$
shows *modify-map* (*modify-map m x g*) *y f* = *modify-map* (*modify-map m y f*) *x g*
by (*rule ext*) (*simp add*: *modify-map-def map-option-case com split*: *option.splits*)

lemma *modify-map-dom* :
dom (*modify-map m p f*) = *dom m*
unfolding *modify-map-def* **by** (*auto simp*: *dom-def*)

lemma *modify-map-None*:
m x = *None* $\Longrightarrow$ *modify-map m x f* = *m*
by (*rule ext*) (*simp add*: *modify-map-def*)

lemma *modify-map-ndom* :
$x \notin$ *dom m* $\Longrightarrow$ *modify-map m x f* = *m*
by (*rule modify-map-None*) *clarsimp*

lemma *modify-map-app*:
(*modify-map m p f*) *q* = (*if p* = *q then map-option f* (*m p*) *else m q*)
unfolding *modify-map-def* **by** *simp*

lemma *modify-map-apply*:
m p = *Some x* $\Longrightarrow$ *modify-map m p f* = *m* ($p \mapsto f\ x$)
by (*simp add*: *modify-map-def*)

lemma *modify-map-com*:
assumes *com*: $\bigwedge x.\ f\ (g\ x) = g\ (f\ x)$
shows *modify-map* (*modify-map m x g*) *y f* = *modify-map* (*modify-map m y f*) *x g*
using *assms* **by** (*auto simp*: *modify-map-def map-option-case split*: *option.splits*)

lemma *modify-map-comp*:
modify-map m x (*f o g*) = *modify-map* (*modify-map m x g*) *x f*
by (*rule ext*) (*simp add*: *modify-map-def option.map-comp*)

lemma *modify-map-exists-eq*:
($\exists$ *cte*. *modify-map m p′ f p*= *Some cte*) = ($\exists$ *cte*. *m p* = *Some cte*)
by (*auto simp*: *modify-map-def split*: *if-splits*)

lemma *modify-map-other*:
$p \neq q \Longrightarrow (\textit{modify-map } m\ p\ f)\ q = (m\ q)$
by (*simp add*: *modify-map-app*)

lemma *modify-map-same*:
$\textit{modify-map } m\ p\ f\ p = \textit{map-option } f\ (m\ p)$
by (*simp add*: *modify-map-app*)

lemma *next-update-is-modify*:
$[\![\ m\ p = \textit{Some } cte';\ cte = f\ cte'\ ]\!] \Longrightarrow (m(p \mapsto cte)) = \textit{modify-map } m\ p\ f$
unfolding *modify-map-def* **by** *simp*

lemma *nat-power-minus-less*:
$a < 2\ \hat{}\ (x - n) \Longrightarrow (a :: nat) < 2\ \hat{}\ x$
by (*erule order-less-le-trans*) *simp*

lemma *neg-rtranclI*:
$[\![\ x \neq y;\ (x,\ y) \notin R^+\ ]\!] \Longrightarrow (x,\ y) \notin R^*$
by (*meson rtranclD*)

lemma *neg-rtrancl-into-trancl*:
$\neg\ (x,\ y) \in R^* \Longrightarrow \neg\ (x,\ y) \in R^+$
by (*erule contrapos-nn*, *erule trancl-into-rtrancl*)

lemma *set-neqI*:
$[\![\ x \in S;\ x \notin S'\ ]\!] \Longrightarrow S \neq S'$
by *clarsimp*

lemma *set-pair-UN*:
$\{x.\ P\ x\} = \textit{UNION}\ \{xa.\ \exists\, xb.\ P\ (xa,\ xb)\}\ (\lambda xa.\ \{xa\} \times \{xb.\ P\ (xa,\ xb)\})$
by *fastforce*

lemma *singleton-elemD*: $S = \{x\} \Longrightarrow x \in S$
by *simp*

lemma *singleton-eqD*: $A = \{x\} \Longrightarrow x \in A$
by *blast*

lemma *ball-ran-fun-updI*:
$[\![\ \forall\, v \in \textit{ran } m.\ P\ v;\ \forall\, v.\ y = \textit{Some } v \longrightarrow P\ v\ ]\!] \Longrightarrow \forall\, v \in \textit{ran}\ (m\ (x := y)).\ P\ v$
by (*auto simp add*: *ran-def*)

lemma *ball-ran-eq*:
$(\forall\, y \in \textit{ran } m.\ P\ y) = (\forall\, x\ y.\ m\ x = \textit{Some } y \longrightarrow P\ y)$
by (*auto simp add*: *ran-def*)

lemma *cart-helper*:
$(\{\} = \{x\} \times S) = (S = \{\})$
by *blast*

lemmas *converse-trancl-induct'* = *converse-trancl-induct* [*consumes 1*, *case-names base step*]

lemma *disjCI2*: $(\neg\ P \Longrightarrow Q) \Longrightarrow P \vee Q$ **by** *blast*

lemma *insert-UNIV* :
insert x UNIV = *UNIV*
by *blast*

lemma *not-singletonE*:
$[\![\ \forall\, p.\ S \neq \{p\};\ S \neq \{\};\ \bigwedge p\ p'.\ [\![\ p \neq p';\ p \in S;\ p' \in S\ ]\!] \Longrightarrow R\ ]\!] \Longrightarrow R$
by *blast*

lemma *not-singleton-oneE*:
$[\![\ \forall\, p.\ S \neq \{p\};\ p \in S;\ \bigwedge p'.\ [\![\ p \neq p';\ p' \in S\ ]\!] \Longrightarrow R\ ]\!] \Longrightarrow R$
using *not-singletonE* **by** *fastforce*

lemma *ball-ran-modify-map-eq*:
$[\![\ \forall\, v.\ m\ x = Some\ v \longrightarrow P\ (f\ v) = P\ v\ ]\!]$
$\Longrightarrow (\forall\, v \in ran\ (modify\text{-}map\ m\ x\ f).\ P\ v) = (\forall\, v \in ran\ m.\ P\ v)$
by (*auto simp*: *modify-map-def ball-ran-eq*)

lemma *disj-imp*: $(P \vee Q) = (\neg P \longrightarrow Q)$ **by** *blast*

lemma *eq-singleton-redux*:
$[\![\ S = \{x\}\ ]\!] \Longrightarrow x \in S$
by *simp*

lemma *if-eq-elem-helperE*:
$[\![\ x \in (if\ P\ then\ S\ else\ S');\ \ [\![\ P;\ \ \ x \in S\ \ ]\!] \Longrightarrow a = b;\ \ [\![\ \neg\ P;\ x \in S'\ ]\!] \Longrightarrow a = c\ ]\!]$
$\Longrightarrow a = (if\ P\ then\ b\ else\ c)$
by *fastforce*

lemma *if-option-Some*:
$((if\ P\ then\ None\ else\ Some\ x) = Some\ y) = (\neg P \wedge x = y)$
by *simp*

lemma *insert-minus-eq*:
$x \notin A \Longrightarrow A - S = (A - (S - \{x\}))$
by *auto*

lemma *modify-map-K-D*:
$modify\text{-}map\ m\ p\ (\lambda x.\ y)\ p' = Some\ v \Longrightarrow (m\ (p \mapsto y))\ p' = Some\ v$
by (*simp add*: *modify-map-def split*: *if-split-asm*)

lemma *tranclE2*:
assumes *trancl*: $(a,\ b) \in r^+$

and *base*: $(a,\ b) \in r \Longrightarrow P$
and *step*: $\bigwedge c.\ [\![(a,\ c) \in r;\ (c,\ b) \in r^+]\!] \Longrightarrow P$
shows P
using *trancl base step*
proof −
note rl = *converse-trancl-induct* [**where** $P = \lambda x.\ x = a \longrightarrow P$]
from *trancl* **have** $a = a \longrightarrow P$
by (*rule rl*, (*iprover intro*: *base step*)+)
thus *?thesis* **by** *simp*
qed

lemmas *tranclE2′* = *tranclE2* [*consumes 1*, *case-names base trancl*]

lemma *weak-imp-cong*:
$[\![\ P = R;\ Q = S\ ]\!] \Longrightarrow (P \longrightarrow Q) = (R \longrightarrow S)$
by *simp*

lemma *Collect-Diff-restrict-simp*:
$T - \{x \in T.\ Q\ x\} = T - \{x.\ Q\ x\}$
by (*auto intro*: *Collect-cong*)

lemma *Collect-Int-pred-eq*:
$\{x \in S.\ P\ x\} \cap \{x \in T.\ P\ x\} = \{x \in (S \cap T).\ P\ x\}$
by (*simp add*: *Collect-conj-eq* [*symmetric*] *conj-comms*)

lemma *Collect-restrict-predR*:
$\{x.\ P\ x\} \cap T = \{\} \Longrightarrow \{x.\ P\ x\} \cap \{x \in T.\ Q\ x\} = \{\}$
by (*fastforce simp*: *disjoint-iff-not-equal*)

lemma *Diff-Un2*:
assumes *emptyad*: $A \cap D = \{\}$
and *emptybc*: $B \cap C = \{\}$
shows $(A \cup B) - (C \cup D) = (A - C) \cup (B - D)$
proof −
have $(A \cup B) - (C \cup D) = (A \cup B - C) \cap (A \cup B - D)$
by (*rule Diff-Un*)
also have $\ldots = ((A - C) \cup B) \cap (A \cup (B - D))$ **using** *emptyad emptybc*
by (*simp add*: *Un-Diff Diff-triv*)
also have $\ldots = (A - C) \cup (B - D)$
proof −
have $(A - C) \cap (A \cup (B - D)) = A - C$ **using** *emptyad emptybc*
by (*metis Diff-Int2 Diff-Int-distrib2 inf-sup-absorb*)
moreover
have $B \cap (A \cup (B - D)) = B - D$ **using** *emptyad emptybc*
by (*metis Int-Diff Un-Diff Un-Diff-Int Un-commute Un-empty-left inf-sup-absorb*)
ultimately show *?thesis*
by (*simp add*: *Int-Un-distrib2*)
qed
finally show *?thesis* **.**

qed

lemma *ballEI*:
$[\![\forall x \in S.\ Q\ x;\ \bigwedge x.\ [\![x \in S;\ Q\ x]\!] \Longrightarrow P\ x]\!] \Longrightarrow \forall x \in S.\ P\ x$
by *auto*

lemma *dom-if-None*:
$dom\ (\lambda x.\ if\ P\ x\ then\ None\ else\ f\ x) = dom\ f - \{x.\ P\ x\}$
by (*simp add*: *dom-def*) *fastforce*

lemma *restrict-map-Some-iff*:
$((m\ |`\ S)\ x = Some\ y) = (m\ x = Some\ y \wedge x \in S)$
by (*cases* $x \in S$, *simp-all*)

lemma *context-case-bools*:
$[\![\bigwedge v.\ P\ v \Longrightarrow R\ v;\ [\![\neg\ P\ v;\ \bigwedge v.\ P\ v \Longrightarrow R\ v]\!] \Longrightarrow R\ v]\!] \Longrightarrow R\ v$
by (*cases* $P\ v$, *simp-all*)

lemma *inj-on-fun-upd-strongerI*:
$[\![inj\text{-}on\ f\ A;\ y \notin f\ `\ (A - \{x\})]\!] \Longrightarrow inj\text{-}on\ (f(x := y))\ A$
by (*fastforce simp*: *inj-on-def*)

lemma *less-handy-casesE*:
$[\![m < n;\ m = 0 \Longrightarrow R;\ \bigwedge m'\ n'.\ [\![n = Suc\ n';\ m = Suc\ m';\ m < n]\!] \Longrightarrow R]\!]$
$\Longrightarrow R$
by (*case-tac n*; *simp*) (*case-tac m*; *simp*)

lemma *subset-drop-Diff-strg*:
$(A \subseteq C) \longrightarrow (A - B \subseteq C)$
by *blast*

lemma *inj-case-bool*:
$inj\ (case\text{-}bool\ a\ b) = (a \neq b)$
by (*auto dest*: *inj-onD*[**where** *x*=*True* **and** *y*=*False*] *intro*: *inj-onI split*: *bool.split-asm*)

lemma *foldl-fun-upd*:
$foldl\ (\lambda s\ r.\ s\ (r := g\ r))\ f\ rs = (\lambda x.\ if\ x \in set\ rs\ then\ g\ x\ else\ f\ x)$
by (*induct rs arbitrary*: *f*) (*auto simp*: *fun-eq-iff*)

lemma *all-rv-choice-fn-eq-pred*:
$[\![\bigwedge rv.\ P\ rv \Longrightarrow \exists fn.\ f\ rv = g\ fn]\!] \Longrightarrow \exists fn.\ \forall rv.\ P\ rv \longrightarrow f\ rv = g\ (fn\ rv)$
apply (*rule-tac x*=$\lambda rv.\ SOME\ h.\ f\ rv = g\ h$ **in** *exI*)
apply (*clarsimp split*: *if-split*)
by (*meson someI-ex*)

lemma *ex-const-function*:
$\exists f.\ \forall s.\ f\ (f'\ s) = v$
by *force*

lemma *if-Const-helper*:
  $If\ P\ (Con\ x)\ (Con\ y) = Con\ (If\ P\ x\ y)$
  by (*simp split*: *if-split*)

lemmas *if-Some-helper* = *if-Const-helper*[**where** *Con=Some*]

lemma *expand-restrict-map-eq*:
  $(m\ |`\ S = m'\ |`\ S) = (\forall x.\ x \in S \longrightarrow m\ x = m'\ x)$
  by (*simp add*: *fun-eq-iff restrict-map-def split*: *if-split*)

lemma *disj-imp-rhs*:
  $(P \Longrightarrow Q) \Longrightarrow (P \lor Q) = Q$
  by *blast*

lemma *remove1-filter*:
  $distinct\ xs \Longrightarrow remove1\ x\ xs = filter\ (\lambda y.\ x \neq y)\ xs$
  by (*induct xs*) (*auto intro*!: *filter-True* [*symmetric*])

lemma *Int-Union-empty*:
  $(\bigwedge x.\ x \in S \Longrightarrow A \cap P\ x = \{\}) \Longrightarrow A \cap (\bigcup x \in S.\ P\ x) = \{\}$
  by *auto*

lemma *UN-Int-empty*:
  $(\bigwedge x.\ x \in S \Longrightarrow P\ x \cap T = \{\}) \Longrightarrow (\bigcup x \in S.\ P\ x) \cap T = \{\}$
  by *auto*

lemma *disjointI*:
  $[\![\bigwedge x\ y.\ [\![\ x \in A;\ y \in B\ ]\!] \Longrightarrow x \neq y\ ]\!] \Longrightarrow A \cap B = \{\}$
  by *auto*

lemma *UN-disjointI*:
  assumes *rl*: $\bigwedge x\ y.\ [\![\ x \in A;\ y \in B\ ]\!] \Longrightarrow P\ x \cap Q\ y = \{\}$
  shows $(\bigcup x \in A.\ P\ x) \cap (\bigcup x \in B.\ Q\ x) = \{\}$
  by (*auto dest*: *rl*)

lemma *UN-set-member*:
  assumes *sub*: $A \subseteq (\bigcup x \in S.\ P\ x)$
  and *nz*: $A \neq \{\}$
  shows $\exists x \in S.\ P\ x \cap A \neq \{\}$
proof −
  from *nz* **obtain** *z* **where** *zA*: $z \in A$ **by** *fastforce*
  with *sub* **obtain** *x* **where** $x \in S$ **and** $z \in P\ x$ **by** *auto*
  hence $P\ x \cap A \neq \{\}$ **using** *zA* **by** *auto*
  thus *?thesis* **using** *sub nz* **by** *auto*
qed

lemma *append-Cons-cases* [*consumes 1*, *case-names pre mid post*]:
  $[\![(x,\ y) \in set\ (as\ @\ b\ \#\ bs);$
    $(x,\ y) \in set\ as \Longrightarrow R;$

$[\![(x,\ y) \notin \mathit{set}\ \mathit{as};\ (x,\ y) \notin \mathit{set}\ \mathit{bs};\ (x,\ y) = b]\!] \Longrightarrow R;$
$(x,\ y) \in \mathit{set}\ \mathit{bs} \Longrightarrow R]\!] \Longrightarrow R$
by *auto*

lemma *cart-singletons*:
$\{a\} \times \{b\} = \{(a,\ b)\}$
by *blast*

lemma *disjoint-subset-neg1*:
$[\![\ B \cap C = \{\};\ A \subseteq B;\ A \neq \{\}\ ]\!] \Longrightarrow \neg\ A \subseteq C$
by *auto*

lemma *disjoint-subset-neg2*:
$[\![\ B \cap C = \{\};\ A \subseteq C;\ A \neq \{\}\ ]\!] \Longrightarrow \neg\ A \subseteq B$
by *auto*

lemma *iffE2*:
$[\![\ P = Q;\ [\![\ P;\ Q\ ]\!] \Longrightarrow R;\ [\![\ \neg\ P;\ \neg\ Q\ ]\!] \Longrightarrow R\ ]\!] \Longrightarrow R$
by *blast*

lemma *list-case-If*:
$(\mathit{case}\ \mathit{xs}\ \mathit{of}\ [] \Rightarrow P\ |\ \text{-} \Rightarrow Q) = (\mathit{if}\ \mathit{xs} = []\ \mathit{then}\ P\ \mathit{else}\ Q)$
by (*rule list.case-eq-if*)

lemma *remove1-Nil-in-set*:
$[\![\ \mathit{remove1}\ x\ \mathit{xs} = [];\ \mathit{xs} \neq []\ ]\!] \Longrightarrow x \in \mathit{set}\ \mathit{xs}$
by (*induct xs*) (*auto split*: *if-split-asm*)

lemma *remove1-empty*:
$(\mathit{remove1}\ v\ \mathit{xs} = []) = (\mathit{xs} = [v] \lor \mathit{xs} = [])$
by (*cases xs*; *simp*)

lemma *set-remove1*:
$x \in \mathit{set}\ (\mathit{remove1}\ y\ \mathit{xs}) \Longrightarrow x \in \mathit{set}\ \mathit{xs}$
by (*induct xs*) (*auto split*: *if-split-asm*)

lemma *If-rearrage*:
$(\mathit{if}\ P\ \mathit{then}\ \mathit{if}\ Q\ \mathit{then}\ x\ \mathit{else}\ y\ \mathit{else}\ z) = (\mathit{if}\ P \land Q\ \mathit{then}\ x\ \mathit{else}\ \mathit{if}\ P\ \mathit{then}\ y\ \mathit{else}\ z)$
by *simp*

lemma *disjI2-strg*:
$Q \longrightarrow (P \lor Q)$
by *simp*

lemma *eq-imp-strg*:
$P\ t \longrightarrow (t = s \longrightarrow P\ s)$
by *clarsimp*

lemma *if-both-strengthen*:

$P \wedge Q \longrightarrow (\textit{if } G \textit{ then } P \textit{ else } Q)$
by *simp*

lemma *if-both-strengthen2*:
$P\ s \wedge Q\ s \longrightarrow (\textit{if } G \textit{ then } P \textit{ else } Q)\ s$
by *simp*

lemma *if-swap*:
$(\textit{if } P \textit{ then } Q \textit{ else } R) = (\textit{if } \neg P \textit{ then } R \textit{ else } Q)$ **by** *simp*

lemma *imp-consequent*:
$P \longrightarrow Q \longrightarrow P$ **by** *simp*

lemma *list-case-helper*:
$xs \neq [] \Longrightarrow \textit{case-list } f\ g\ xs = g\ (\textit{hd } xs)\ (\textit{tl } xs)$
by (*cases xs*, *simp-all*)

lemma *list-cons-rewrite*:
$(\forall x\ xs.\ L = x \# xs \longrightarrow P\ x\ xs) = (L \neq [] \longrightarrow P\ (\textit{hd } L)\ (\textit{tl } L))$
by (*auto simp*: *neq-Nil-conv*)

lemma *list-not-Nil-manip*:
$[\![\ xs = y \# ys;\ \textit{case } xs \textit{ of } [] \Rightarrow \textit{False} \mid (y \# ys) \Rightarrow P\ y\ ys\ ]\!] \Longrightarrow P\ y\ ys$
by *simp*

lemma *ran-ball-triv*:
$\bigwedge P\ m\ S.\ [\![\ \forall x \in (\textit{ran } S).\ P\ x\ ;\ m \in (\textit{ran } S)\ ]\!] \Longrightarrow P\ m$
by *blast*

lemma *singleton-tuple-cartesian*:
$(\{(a, b)\} = S \times T) = (\{a\} = S \wedge \{b\} = T)$
$(S \times T = \{(a, b)\}) = (\{a\} = S \wedge \{b\} = T)$
by *blast+*

lemma *strengthen-ignore-if*:
$A\ s \wedge B\ s \longrightarrow (\textit{if } P \textit{ then } A \textit{ else } B)\ s$
by *clarsimp*

lemma *case-sum-True* :
$(\textit{case } r \textit{ of Inl } a \Rightarrow \textit{True} \mid \textit{Inr } b \Rightarrow f\ b) = (\forall b.\ r = \textit{Inr } b \longrightarrow f\ b)$
by (*cases r*) *auto*

lemma *sym-ex-elim*:
$F\ x = y \Longrightarrow \exists x.\ y = F\ x$
by *auto*

lemma *tl-drop-1* :
$\textit{tl } xs = \textit{drop } 1\ xs$
by (*simp add*: *drop-Suc*)

lemma *upt-lhs-sub-map*:
$[x\ ..<\ y] = map\ ((+)\ x)\ [0\ ..<\ y - x]$
by (*induct y*) (*auto simp*: *Suc-diff-le*)

lemma *upto-0-to-4*:
$[0..<4] = 0\ \#\ [1..<4]$
by (*subst upt-rec*) *simp*

lemma *disjEI*:
$[\![\ P \lor Q;\ P \Longrightarrow R;\ Q \Longrightarrow S\ ]\!]$
$\Longrightarrow R \lor S$
by *fastforce*

lemma *dom-fun-upd2*:
$s\ x = Some\ z \Longrightarrow dom\ (s\ (x \mapsto y)) = dom\ s$
by (*simp add*: *insert-absorb domI*)

lemma *foldl-True* :
$foldl\ (\lor)\ True\ bs$
by (*induct bs*) *auto*

lemma *image-set-comp*:
$f\ `\ \{g\ x\ |\ x.\ Q\ x\} = (f \circ g)\ `\ \{x.\ Q\ x\}$
by *fastforce*

lemma *mutual-exE*:
$[\![\ \exists x.\ P\ x;\ \bigwedge x.\ P\ x \Longrightarrow Q\ x\ ]\!] \Longrightarrow \exists x.\ Q\ x$
by *blast*

lemma *nat-diff-eq*:
fixes $x :: nat$
shows $[\![\ x - y = x - z;\ y < x]\!] \Longrightarrow y = z$
by *arith*

lemma *comp-upd-simp*:
$(f \circ (g\ (x := y))) = ((f \circ g)\ (x := f\ y))$
by (*rule fun-upd-comp*)

lemma *dom-option-map*:
$dom\ (map\text{-}option\ f\ o\ m) = dom\ m$
by (*rule dom-map-option-comp*)

lemma *drop-imp*:
$P \Longrightarrow (A \longrightarrow P) \land (B \longrightarrow P)$ **by** *blast*

lemma *inj-on-fun-updI2*:
$[\![\ inj\text{-}on\ f\ A;\ y \notin f\ `\ (A - \{x\})\ ]\!] \Longrightarrow inj\text{-}on\ (f(x := y))\ A$
by (*rule inj-on-fun-upd-strongerI*)

lemma *inj-on-fun-upd-elsewhere*:
$x \notin S \Longrightarrow inj\text{-}on\ (f\ (x := y))\ S = inj\text{-}on\ f\ S$
by (*simp add*: *inj-on-def*) *blast*

lemma *not-Some-eq-tuple*:
$(\forall\, y\ z.\ x \neq Some\ (y, z)) = (x = None)$
by (*cases x*, *simp-all*)

lemma *ran-option-map*:
$ran\ (map\text{-}option\ f\ o\ m) = f\ `\ ran\ m$
by (*auto simp add*: *ran-def*)

lemma *All-less-Ball*:
$(\forall\, x < n.\ P\ x) = (\forall\, x \in \{..< n\}.\ P\ x)$
by *fastforce*

lemma *Int-image-empty*:
$[\![\bigwedge x\ y.\ f\ x \neq g\ y\ ]\!]$
$\Longrightarrow f\ `\ S \cap g\ `\ T = \{\}$
by *auto*

lemma *Max-prop*:
$[\![Max\ S \in S \Longrightarrow P\ (Max\ S); (S :: ('a :: \{finite, linorder\})\ set) \neq \{\}\ ]\!] \Longrightarrow P\ (Max\ S)$
by *auto*

lemma *Min-prop*:
$[\![Min\ S \in S \Longrightarrow P\ (Min\ S); (S :: ('a :: \{finite, linorder\})\ set) \neq \{\}\ ]\!] \Longrightarrow P\ (Min\ S)$
by *auto*

lemma *findSomeD*:
$find\ P\ xs = Some\ x \Longrightarrow P\ x \wedge x \in set\ xs$
by (*induct xs*) (*auto split*: *if-split-asm*)

lemma *findNoneD*:
$find\ P\ xs = None \Longrightarrow \forall\, x \in set\ xs.\ \neg P\ x$
by (*induct xs*) (*auto split*: *if-split-asm*)

lemma *dom-upd*:
$dom\ (\lambda x.\ if\ x = y\ then\ None\ else\ f\ x) = dom\ f - \{y\}$
by (*rule set-eqI*) (*auto split*: *if-split-asm*)

definition
$is\text{-}inv :: ('a \rightharpoonup 'b) \Rightarrow ('b \rightharpoonup 'a) \Rightarrow bool$ **where**
$is\text{-}inv\ f\ g \equiv ran\ f = dom\ g \wedge (\forall\, x\ y.\ f\ x = Some\ y \longrightarrow g\ y = Some\ x)$

lemma *is-inv-NoneD*:
  assumes $g\ x = None$
  assumes *is-inv f g*
  shows $x \notin ran\ f$
proof −
  from *assms*
  have $x \notin dom\ g$ **by** (*auto simp*: *ran-def*)
  moreover
  from *assms*
  have $ran\ f = dom\ g$
    by (*simp add*: *is-inv-def*)
  ultimately
  show *?thesis* **by** *simp*
qed

lemma *is-inv-SomeD*:
  $[\![\ f\ x = Some\ y;\ is\text{-}inv\ f\ g\ ]\!] \Longrightarrow g\ y = Some\ x$
  by (*simp add*: *is-inv-def*)

lemma *is-inv-com*:
  $is\text{-}inv\ f\ g \Longrightarrow is\text{-}inv\ g\ f$
  apply (*unfold is-inv-def*)
  apply *safe*
    apply (*clarsimp simp*: *ran-def dom-def set-eq-iff*)
    apply (*erule-tac x=a* **in** *allE*)
    apply *clarsimp*
   apply (*clarsimp simp*: *ran-def dom-def set-eq-iff*)
   apply *blast*
  apply (*clarsimp simp*: *ran-def dom-def set-eq-iff*)
  apply (*erule-tac x=x* **in** *allE*)
  apply *clarsimp*
  done

lemma *is-inv-inj*:
  $is\text{-}inv\ f\ g \Longrightarrow inj\text{-}on\ f\ (dom\ f)$
  apply (*frule is-inv-com*)
  apply (*clarsimp simp*: *inj-on-def*)
  apply (*drule* (*1*) *is-inv-SomeD*)
  apply (*auto dest*: *is-inv-SomeD*)
  done

lemma *ran-upd'*:
  $[\![inj\text{-}on\ f\ (dom\ f);\ f\ y = Some\ z]\!] \Longrightarrow ran\ (f\ (y := None)) = ran\ f - \{z\}$
  by (*force simp*: *ran-def inj-on-def dom-def intro*!: *set-eqI*)

lemma *is-inv-None-upd*:
  $[\![\ is\text{-}inv\ f\ g;\ g\ x = Some\ y]\!] \Longrightarrow is\text{-}inv\ (f(y := None))\ (g(x := None))$
  apply (*subst is-inv-def*)
  apply (*clarsimp simp*: *dom-upd*)

apply (*drule is-inv-SomeD*, *erule is-inv-com*)
apply (*frule is-inv-inj*)
apply (*auto simp*: *ran-upd′ is-inv-def dest*: *is-inv-SomeD is-inv-inj*)
done

lemma *is-inv-inj2*:
$is\text{-}inv\ f\ g \Longrightarrow inj\text{-}on\ g\ (dom\ g)$
using *is-inv-com is-inv-inj* **by** *blast*

lemma *range-convergence1*:
$[\![\ \forall z.\ x < z \wedge z \leq y \longrightarrow P\ z;\ \forall z > y.\ P\ (z :: 'a :: linorder)\ ]\!] \Longrightarrow \forall z > x.\ P\ z$
using *not-le* **by** *blast*

lemma *range-convergence2*:
$[\![\ \forall z.\ x < z \wedge z \leq y \longrightarrow P\ z;\ \forall z.\ z > y \wedge z < w \longrightarrow P\ (z :: 'a :: linorder)\ ]\!]$
$\Longrightarrow \forall z.\ z > x \wedge z < w \longrightarrow P\ z$
using *range-convergence1*[**where** $P{=}\lambda z.\ z < w \longrightarrow P\ z$ **and** $x{=}x$ **and** $y{=}y$]
by *auto*

lemma *zip-upt-Cons*:
$a < b \Longrightarrow zip\ [a\ ..{<}\ b]\ (x\ \#\ xs) = (a,\ x)\ \#\ zip\ [Suc\ a\ ..{<}\ b]\ xs$
by (*simp add*: *upt-conv-Cons*)

lemma *map-comp-eq*:
$f \circ_m g = case\text{-}option\ None\ f \circ g$
apply (*rule ext*)
apply (*case-tac g x*)
by *auto*

lemma *dom-If-Some*:
$dom\ (\lambda x.\ if\ x \in S\ then\ Some\ v\ else\ f\ x) = (S \cup dom\ f)$
by (*auto split*: *if-split*)

lemma *foldl-fun-upd-const*:
$foldl\ (\lambda s\ x.\ s(f\ x := v))\ s\ xs$
$= (\lambda x.\ if\ x \in f\ `\ set\ xs\ then\ v\ else\ s\ x)$
by (*induct xs arbitrary*: *s*) *auto*

lemma *foldl-id*:
$foldl\ (\lambda s\ x.\ s)\ s\ xs = s$
by (*induct xs*) *auto*

lemma *SucSucMinus*: $2 \leq n \Longrightarrow Suc\ (Suc\ (n - 2)) = n$ **by** *arith*

lemma *ball-to-all*:
$(\bigwedge x.\ (x \in A) = (P\ x)) \Longrightarrow (\forall x \in A.\ B\ x) = (\forall x.\ P\ x \longrightarrow B\ x)$
by *blast*

lemma *case-option-If*:

case-option P $(\lambda x.\ Q)$ v = (*if* v = *None then* P *else* Q)
by *clarsimp*

lemma *case-option-If2*:
case-option P Q v = *If* ($v \neq$ *None*) (Q (*the* v)) P
by (*simp split*: *option.split*)

lemma *if3-fold*:
(*if* P *then* x *else if* Q *then* y *else* x) = (*if* $P \vee \neg$ Q *then* x *else* y)
by *simp*

lemma *rtrancl-insert*:
assumes *x-new*: $\bigwedge y.\ (x,y) \notin R$
shows $R\hat{}*$ ‘‘ *insert* x S = *insert* x ($R\hat{}*$ ‘‘ S)
proof −
have $R\hat{}*$ ‘‘ *insert* x S = $R\hat{}*$ ‘‘ $(\{x\} \cup S)$ **by** *simp*
also
have $R\hat{}*$ ‘‘ $(\{x\} \cup S)$ = $R\hat{}*$ ‘‘ $\{x\} \cup R\hat{}*$ ‘‘ S
by (*subst Image-Un*) *simp*
also
have $R\hat{}*$ ‘‘ $\{x\} = \{x\}$
by (*meson Image-closed-trancl Image-singleton-iff subsetI x-new*)
finally
show *?thesis* **by** *simp*
qed

lemma *ran-del-subset*:
$y \in$ *ran* (f (x := *None*)) $\Longrightarrow$ $y \in$ *ran* f
by (*auto simp*: *ran-def split*: *if-split-asm*)

lemma *trancl-sub-lift*:
assumes *sub*: $\bigwedge p\ p'.\ (p,p') \in r \Longrightarrow (p,p') \in r'$
shows $(p,p') \in r\hat{}+ \Longrightarrow (p,p') \in r'\hat{}+$
by (*fastforce intro*: *trancl-mono sub*)

lemma *trancl-step-lift*:
assumes *x-step*: $\bigwedge p\ p'.\ (p,p') \in r' \Longrightarrow (p,p') \in r \vee (p = x \wedge p' = y)$
assumes *y-new*: $\bigwedge p'.\ \neg(y,p') \in r$
shows $(p,p') \in r'\hat{}+ \Longrightarrow (p,p') \in r\hat{}+ \vee ((p,x) \in r\hat{}+ \wedge p' = y) \vee (p = x \wedge p'$
$= y)$
apply (*erule trancl-induct*)
apply (*drule x-step*)
apply *fastforce*
apply (*erule disjE*)
apply (*drule x-step*)
apply (*erule disjE*)
apply (*drule trancl-trans*, *drule r-into-trancl*, *assumption*)
apply *blast*
apply *fastforce*

apply (*fastforce simp*: *y-new dest*: *x-step*)
done

lemma *rtrancl-simulate-weak*:
assumes *r*: $(x,z) \in R^*$
assumes *s*: $\bigwedge y.\ (x,y) \in R \Longrightarrow (y,z) \in R^* \Longrightarrow (x,y) \in R' \wedge (y,z) \in R'^*$
shows $(x,z) \in R'^*$
apply (*rule converse-rtranclE*[*OF r*])
apply *simp*
apply (*frule* (*1*) *s*)
apply *clarsimp*
by (*rule converse-rtrancl-into-rtrancl*)

lemma *list-case-If2*:
case-list f g xs = *If* (*xs* = []) *f* (*g* (*hd xs*) (*tl xs*))
by (*simp split*: *list.split*)

lemma *length-ineq-not-Nil*:
$length\ xs > n \Longrightarrow xs \neq []$
$length\ xs \geq n \Longrightarrow n \neq 0 \longrightarrow xs \neq []$
$\neg\ length\ xs < n \Longrightarrow n \neq 0 \longrightarrow xs \neq []$
$\neg\ length\ xs \leq n \Longrightarrow xs \neq []$
by *auto*

lemma *numeral-eqs*:
2 = *Suc* (*Suc 0*)
3 = *Suc* (*Suc* (*Suc 0*))
4 = *Suc* (*Suc* (*Suc* (*Suc 0*)))
5 = *Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))
6 = *Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))
by *simp+*

lemma *psubset-singleton*:
$(S \subset \{x\}) = (S = \{\})$
by *blast*

lemma *length-takeWhile-ge*:
$length\ (takeWhile\ f\ xs) = n \Longrightarrow length\ xs = n \vee (length\ xs > n \wedge \neg\ f\ (xs\ !\ n))$
by (*induct xs arbitrary*: *n*) (*auto split*: *if-split-asm*)

lemma *length-takeWhile-le*:
$\neg\ f\ (xs\ !\ n) \Longrightarrow length\ (takeWhile\ f\ xs) \leq n$
by (*induct xs arbitrary*: *n*; *simp*) (*case-tac n*; *simp*)

lemma *length-takeWhile-gt*:
$n < length\ (takeWhile\ f\ xs)$
$\Longrightarrow (\exists\ ys\ zs.\ length\ ys = Suc\ n \wedge xs = ys\ @\ zs \wedge takeWhile\ f\ xs = ys\ @\ takeWhile\ f\ zs)$
apply (*induct xs arbitrary*: *n*; *simp split*: *if-split-asm*)

apply (*case-tac n*; *simp*)
 apply (*rule-tac x*=[*a*] **in** *exI*)
 apply *simp*
apply (*erule meta-allE*, *drule*(*1*) *meta-mp*)
apply *clarsimp*
apply (*rule-tac x*=*a* # *ys* **in** *exI*)
apply *simp*
done

lemma *hd-drop-conv-nth2*:
$n < length\ xs \Longrightarrow hd\ (drop\ n\ xs) = xs\ !\ n$
by (*rule hd-drop-conv-nth*) *clarsimp*

lemma *map-upt-eq-vals-D*:
$[\![\ map\ f\ [0\ ..{<}\ n] = ys;\ m < length\ ys\ ]\!] \Longrightarrow f\ m = ys\ !\ m$
by *clarsimp*

lemma *length-le-helper*:
$[\![\ n \leq length\ xs;\ n \neq 0\ ]\!] \Longrightarrow xs \neq [] \wedge n - 1 \leq length\ (tl\ xs)$
by (*cases xs*, *simp-all*)

lemma *all-ex-eq-helper*:
$(\forall v.\ (\exists v'.\ v = f\ v' \wedge P\ v\ v') \longrightarrow Q\ v)$
$\quad = (\forall v'.\ P\ (f\ v')\ v' \longrightarrow Q\ (f\ v'))$
by *auto*

lemma *nat-less-cases'*:
$(x{::}nat) < y \Longrightarrow x = y - 1 \vee x < y - 1$
by *auto*

lemma *filter-to-shorter-upto*:
$n \leq m \Longrightarrow filter\ (\lambda x.\ x < n)\ [0\ ..{<}\ m] = [0\ ..{<}\ n]$
by (*induct m*) (*auto elim*: *le-SucE*)

lemma *in-emptyE*: $[\![\ A = \{\};\ x \in A\ ]\!] \Longrightarrow P$ **by** *blast*

lemma *Ball-emptyI*:
$S = \{\} \Longrightarrow (\forall x \in S.\ P\ x)$
by *simp*

lemma *allfEI*:
$[\![\ \forall x.\ P\ x;\ \bigwedge x.\ P\ (f\ x) \Longrightarrow Q\ x\ ]\!] \Longrightarrow \forall x.\ Q\ x$
by *fastforce*

lemma *cart-singleton-empty2*:
$(\{x\} \times S = \{\}) = (S = \{\})$
$(\{\} = S \times \{e\}) = (S = \{\})$
by *auto*

lemma *cases-simp-conj*:
$((P \longrightarrow Q) \wedge (\neg\ P \longrightarrow Q) \wedge R) = (Q \wedge R)$
by *fastforce*

lemma *domE* :
$[\![\ x \in dom\ m;\ \bigwedge r.\ [\![m\ x = Some\ r]\!] \Longrightarrow P\ ]\!] \Longrightarrow P$
by *clarsimp*

lemma *dom-eqD*:
$[\![\ f\ x = Some\ v;\ dom\ f = S\ ]\!] \Longrightarrow x \in S$
by *clarsimp*

lemma *exception-set-finite-1*:
$finite\ \{x.\ P\ x\} \Longrightarrow finite\ \{x.\ (x = y \longrightarrow Q\ x) \wedge P\ x\}$
by (*simp add*: *Collect-conj-eq*)

lemma *exception-set-finite-2*:
$finite\ \{x.\ P\ x\} \Longrightarrow finite\ \{x.\ x \neq y \longrightarrow P\ x\}$
by (*simp add*: *imp-conv-disj*)

lemmas *exception-set-finite* = *exception-set-finite-1 exception-set-finite-2*

lemma *exfEI*:
$[\![\ \exists x.\ P\ x;\ \bigwedge x.\ P\ x \Longrightarrow Q\ (f\ x)\ ]\!] \Longrightarrow \exists x.\ Q\ x$
by *fastforce*

lemma *Collect-int-vars*:
$\{s.\ P\ rv\ s\} \cap \{s.\ rv = xf\ s\} = \{s.\ P\ (xf\ s)\ s\} \cap \{s.\ rv = xf\ s\}$
by *auto*

lemma *if-0-1-eq*:
$((if\ P\ then\ 1\ else\ 0) = (case\ Q\ of\ True \Rightarrow of\text{-}nat\ 1\ |\ False \Rightarrow of\text{-}nat\ 0)) = (P = Q)$
by (*simp split*: *if-split bool.split*)

lemma *modify-map-exists-cte* :
$(\exists cte.\ modify\text{-}map\ m\ p\ f\ p' = Some\ cte) = (\exists cte.\ m\ p' = Some\ cte)$
by (*simp add*: *modify-map-def*)

lemma *dom-eqI*:
assumes *c1*: $\bigwedge x\ y.\ P\ x = Some\ y \Longrightarrow \exists y.\ Q\ x = Some\ y$
and *c2*: $\bigwedge x\ y.\ Q\ x = Some\ y \Longrightarrow \exists y.\ P\ x = Some\ y$
shows $dom\ P = dom\ Q$
unfolding *dom-def* **by** (*auto simp*: *c1 c2*)

lemma *dvd-reduce-multiple*:
fixes $k :: nat$
shows $(k\ dvd\ k * m + n) = (k\ dvd\ n)$
by (*induct m*) (*auto simp*: *add-ac*)

lemma *image-iff2*:
$inj\ f \Longrightarrow f\ x \in f\ `\ S = (x \in S)$
by (*rule inj-image-mem-iff*)

lemma *map-comp-restrict-map-Some-iff*:
$((g \circ_m (m\ |`\ S))\ x = Some\ y) = ((g \circ_m m)\ x = Some\ y \wedge x \in S)$
by (*auto simp add*: *map-comp-Some-iff restrict-map-Some-iff*)

lemma *range-subsetD*:
fixes $a :: 'a :: order$
shows $[\![\ \{a..b\} \subseteq \{c..d\};\ a \leq b\ ]\!] \Longrightarrow c \leq a \wedge b \leq d$
by *simp*

lemma *case-option-dom*:
$(case\ f\ x\ of\ None \Rightarrow a\ |\ Some\ v \Rightarrow b\ v) = (if\ x \in dom\ f\ then\ b\ (the\ (f\ x))\ else\ a)$
by (*auto split*: *option.split*)

lemma *contrapos-imp*:
$P \longrightarrow Q \Longrightarrow \neg\ Q \longrightarrow \neg\ P$
by *clarsimp*

lemma *filter-eq-If*:
$distinct\ xs \Longrightarrow filter\ (\lambda v.\ v = x)\ xs = (if\ x \in set\ xs\ then\ [x]\ else\ [])$
by (*induct xs*) *auto*

lemma (**in** *semigroup-add*) *foldl-assoc*:
shows $foldl\ (+)\ (x{+}y)\ zs = x + (foldl\ (+)\ y\ zs)$
by (*induct zs arbitrary*: *y*) (*simp-all add*:*add.assoc*)

lemma (**in** *monoid-add*) *foldl-absorb0*:
shows $x + (foldl\ (+)\ 0\ zs) = foldl\ (+)\ x\ zs$
by (*induct zs*) (*simp-all add*:*foldl-assoc*)

lemma *foldl-conv-concat*:
$foldl\ (@)\ xs\ xss = xs\ @\ concat\ xss$
proof (*induct xss arbitrary*: *xs*)
case *Nil* **show** *?case* **by** *simp*
next
interpret *monoid-add* (@) [] **proof qed** *simp-all*
case *Cons* **then show** *?case* **by** (*simp add*: *foldl-absorb0*)
qed

lemma *foldl-concat-concat*:
$foldl\ (@)\ []\ (xs\ @\ ys) = foldl\ (@)\ []\ xs\ @\ foldl\ (@)\ []\ ys$
by (*simp add*: *foldl-conv-concat*)

lemma *foldl-does-nothing*:
$[\![\ \bigwedge x.\ x \in set\ xs \Longrightarrow f\ s\ x = s\ ]\!] \Longrightarrow foldl\ f\ s\ xs = s$

by (*induct xs*) *auto*

lemma *foldl-use-filter*:
$\llbracket \bigwedge v\ x.\ \llbracket \neg\ g\ x;\ x \in set\ xs\ \rrbracket \Longrightarrow f\ v\ x = v\ \rrbracket \Longrightarrow foldl\ f\ v\ xs = foldl\ f\ v\ (filter\ g\ xs)$
by (*induct xs arbitrary*: *v*) *auto*

lemma *map-comp-update-lift*:
assumes *fv*: $f\ v = Some\ v'$
shows $(f \circ_m (g(ptr \mapsto v))) = ((f \circ_m g)(ptr \mapsto v'))$
by (*simp add*: *fv map-comp-update*)

lemma *restrict-map-cong*:
assumes *sv*: $S = S'$
and *rl*: $\bigwedge p.\ p \in S' \Longrightarrow mp\ p = mp'\ p$
shows $mp\ |`\ S = mp'\ |`\ S'$
using *expand-restrict-map-eq rl sv* **by** *auto*

lemma *case-option-over-if*:
case-option P Q (*if G then None else Some v*)
= (*if G then P else Q v*)
case-option P Q (*if G then Some v else None*)
= (*if G then Q v else P*)
by (*simp split*: *if-split*)+

lemma *map-length-cong*:
$\llbracket length\ xs = length\ ys;\ \bigwedge x\ y.\ (x,\ y) \in set\ (zip\ xs\ ys) \Longrightarrow f\ x = g\ y\ \rrbracket$
$\Longrightarrow map\ f\ xs = map\ g\ ys$
apply *atomize*
apply (*erule rev-mp*, *erule list-induct2*)
apply *auto*
done

lemma *take-min-len*:
take (*min* (*length xs*) *n*) *xs* = *take n xs*
by (*simp add*: *min-def*)

lemmas *interval-empty* = *atLeastatMost-empty-iff*

lemma *fold-and-false*[*simp*]:
$\neg(fold\ (\wedge)\ xs\ False)$
apply *clarsimp*
apply (*induct xs*)
apply *simp*
apply *simp*
done

lemma *fold-and-true*:
$fold\ (\wedge)\ xs\ True \Longrightarrow \forall\, i < length\ xs.\ xs\ !\ i$

apply *clarsimp*
apply (*induct xs*)
apply *simp*
apply (*case-tac i* = *0*; *simp*)
apply (*case-tac a*; *simp*)
apply (*case-tac a*; *simp*)
done

lemma *fold-or-true*[*simp*]:
fold (∨) *xs True*
by (*induct xs*, *simp*+)

lemma *fold-or-false*:
¬(*fold* (∨) *xs False*) ⟹ ∀ *i* < *length xs*. ¬(*xs* ! *i*)
apply (*induct xs*, *simp*+)
apply (*case-tac a*, *simp*+)
apply (*rule allI*, *case-tac i* = *0*, *simp*+)
done

12 Take, drop, zip, list$_{a}lletcrules$

method *two-induct* **for** *xs ys* =
((*induct xs arbitrary*: *ys*; *simp?*), (*case-tac ys*; *simp*)*?*)

lemma *map-fst-zip-prefix*:
map fst (*zip xs ys*) ≤ *xs*
by (*two-induct xs ys*)

lemma *map-snd-zip-prefix*:
map snd (*zip xs ys*) ≤ *ys*
by (*two-induct xs ys*)

lemma *nth-upt-0* [*simp*]:
i < *length xs* ⟹ [*0*..<*length xs*] ! *i* = *i*
by *simp*

lemma *take-insert-nth*:
i < *length xs*⟹ *insert* (*xs* ! *i*) (*set* (*take i xs*)) = *set* (*take* (*Suc i*) *xs*)
by (*subst take-Suc-conv-app-nth*, *assumption*, *fastforce*)

lemma *zip-take-drop*:
⟦*n* < *length xs*; *length ys* = *length xs*⟧ ⟹
zip xs (*take n ys* @ *a* # *drop* (*Suc n*) *ys*) =
zip (*take n xs*) (*take n ys*) @ (*xs* ! *n*, *a*) # *zip* (*drop* (*Suc n*) *xs*) (*drop* (*Suc n*) *ys*)
by (*subst id-take-nth-drop*, *assumption*, *simp*)

lemma *take-nth-distinct*:
⟦*distinct xs*; *n* < *length xs*; *xs* ! *n* ∈ *set* (*take n xs*)⟧ ⟹ *False*

by (*fastforce simp*: *distinct-conv-nth in-set-conv-nth*)

lemma *take-drop-append*:
drop a xs = *take b* (*drop a xs*) @ *drop* (*a* + *b*) *xs*
by (*metis append-take-drop-id drop-drop add.commute*)

lemma *drop-take-drop*:
drop a (*take* (*b* + *a*) *xs*) @ *drop* (*b* + *a*) *xs* = *drop a xs*
by (*metis add.commute take-drop take-drop-append*)

lemma *not-prefixI*:
⟦ *xs* ≠ *ys*; *length xs* = *length ys*⟧ ⟹ ¬ *xs* ≤ *ys*
by (*auto elim*: *prefixE*)

lemma *map-fst-zip'*:
length xs ≤ *length ys* ⟹ *map fst* (*zip xs ys*) = *xs*
by (*metis length-map length-zip map-fst-zip-prefix min-absorb1 not-prefixI*)

lemma *zip-take-triv*:
n ≥ *length bs* ⟹ *zip* (*take n as*) *bs* = *zip as bs*
apply (*induct bs arbitrary*: *n as*; *simp*)
apply (*case-tac n*; *simp*)
apply (*case-tac as*; *simp*)
done

lemma *zip-take-triv2*:
length as ≤ *n* ⟹ *zip as* (*take n bs*) = *zip as bs*
apply (*induct as arbitrary*: *n bs*; *simp*)
apply (*case-tac n*; *simp*)
apply (*case-tac bs*; *simp*)
done

lemma *zip-take-length*:
zip xs (*take* (*length xs*) *ys*) = *zip xs ys*
by (*metis order-refl zip-take-triv2*)

lemma *zip-singleton*:
ys ≠ [] ⟹ *zip* [*a*] *ys* = [(*a*, *ys* ! *0*)]
by (*case-tac ys*, *simp-all*)

lemma *zip-append-singleton*:
⟦*i* = *length xs*; *length xs* < *length ys*⟧ ⟹ *zip* (*xs* @ [*a*]) *ys* = (*zip xs ys*) @ [(*a*,*ys*
! *i*)]
by (*induct xs*; *case-tac ys*; *simp*)
(*clarsimp simp*: *zip-append1 zip-take-length zip-singleton*)

lemma *ran-map-of-zip*:
⟦*length xs* = *length ys*; *distinct xs*⟧ ⟹ *ran* (*map-of* (*zip xs ys*)) = *set ys*
by (*induct rule*: *list-induct2*) *auto*

lemma *ranE*:
  $\llbracket\ v \in ran\ f;\ \bigwedge x.\ f\ x = Some\ v \Longrightarrow R\rrbracket \Longrightarrow R$
  by (*auto simp*: *ran-def*)

lemma *ran-map-option-restrict-eq*:
  $\llbracket\ x \in ran\ (map\text{-}option\ f\ o\ g);\ x \notin ran\ (map\text{-}option\ f\ o\ (g\ |`\ (-\ \{y\})))\ \rrbracket$
  $\Longrightarrow \exists\, v.\ g\ y = Some\ v \wedge f\ v = x$
  apply (*clarsimp simp*: *elim*!: *ranE*)
  apply (*rename-tac w z*)
  apply (*case-tac w = y*)
   apply *clarsimp*
  apply (*erule notE*, *rule-tac a=w* **in** *ranI*)
  apply (*simp add*: *restrict-map-def*)
  done

lemma *map-of-zip-range*:
  $\llbracket length\ xs = length\ ys;\ distinct\ xs\rrbracket \Longrightarrow (\lambda x.\ (the\ (map\text{-}of\ (zip\ xs\ ys)\ x)))\ `\ set\ xs = set\ ys$
  apply (*clarsimp simp*: *image-def*)
  apply (*subst ran-map-of-zip* [*symmetric*, **where** *xs=xs* **and** *ys=ys*]; *simp?*)
  apply (*clarsimp simp*: *ran-def*)
  apply (*rule equalityI*)
   apply *clarsimp*
   apply (*rename-tac x*)
   apply (*frule-tac x=x* **in** *map-of-zip-is-Some*; *fastforce*)
  apply (*clarsimp simp*: *set-zip*)
  by (*metis domI dom-map-of-zip nth-mem ranE ran-map-of-zip option.sel*)

lemma *map-zip-fst*:
  $length\ xs = length\ ys \Longrightarrow map\ (\lambda(x,\ y).\ f\ x)\ (zip\ xs\ ys) = map\ f\ xs$
  by (*two-induct xs ys*)

lemma *map-zip-fst'*:
  $length\ xs \le length\ ys \Longrightarrow map\ (\lambda(x,\ y).\ f\ x)\ (zip\ xs\ ys) = map\ f\ xs$
  by (*metis length-map map-fst-zip' map-zip-fst zip-map-fst-snd*)

lemma *map-zip-snd*:
  $length\ xs = length\ ys \Longrightarrow map\ (\lambda(x,\ y).\ f\ y)\ (zip\ xs\ ys) = map\ f\ ys$
  by (*two-induct xs ys*)

lemma *map-zip-snd'*:
  $length\ ys \le length\ xs \Longrightarrow map\ (\lambda(x,\ y).\ f\ y)\ (zip\ xs\ ys) = map\ f\ ys$
  by (*two-induct xs ys*)

lemma *map-of-zip-tuple-in*:
  $\llbracket(x,\ y) \in set\ (zip\ xs\ ys);\ distinct\ xs\rrbracket \Longrightarrow map\text{-}of\ (zip\ xs\ ys)\ x = Some\ y$
  by (*two-induct xs ys*) (*auto intro*: *in-set-zipE*)

lemma *in-set-zip1*:
$(x,\ y) \in set\ (zip\ xs\ ys) \Longrightarrow x \in set\ xs$
by (*erule in-set-zipE*)

lemma *in-set-zip2*:
$(x,\ y) \in set\ (zip\ xs\ ys) \Longrightarrow y \in set\ ys$
by (*erule in-set-zipE*)

lemma *map-zip-snd-take*:
$map\ (\lambda(x,\ y).\ f\ y)\ (zip\ xs\ ys) = map\ f\ (take\ (length\ xs)\ ys)$
apply (*subst map-zip-snd′* [*symmetric*, **where** *xs*=*xs* **and** *ys*=*take* (*length xs*) *ys*], *simp*)
apply (*subst zip-take-length* [*symmetric*], *simp*)
done

lemma *map-of-zip-is-index*:
$[\![length\ xs = length\ ys;\ x \in set\ xs]\!] \Longrightarrow \exists\, i.\ (map\text{-}of\ (zip\ xs\ ys))\ x = Some\ (ys\ !\ i)$
apply (*induct rule*: *list-induct2*; *simp*)
apply (*rule conjI*; *clarsimp*)
apply (*metis nth-Cons-0*)
apply (*metis nth-Cons-Suc*)
done

lemma *map-of-zip-take-update*:
$[\![i < length\ xs;\ length\ xs \le length\ ys;\ distinct\ xs]\!]$
$\Longrightarrow map\text{-}of\ (zip\ (take\ i\ xs)\ ys)(xs\ !\ i \mapsto (ys\ !\ i)) = map\text{-}of\ (zip\ (take\ (Suc\ i)\ xs)\ ys)$
apply (*rule ext*, *rename-tac x*)
apply (*case-tac x*=*xs* ! *i*; *clarsimp*)
apply (*rule map-of-is-SomeI*[*symmetric*])
apply (*simp add*: *map-fst-zip′*)
apply (*force simp add*: *set-zip*)
apply (*clarsimp simp*: *take-Suc-conv-app-nth zip-append-singleton map-add-def split*: *option.splits*)
done

lemma *map-of-zip-is-Some′*:
$length\ xs \le length\ ys \Longrightarrow (x \in set\ xs) = (\exists\, y.\ map\text{-}of\ (zip\ xs\ ys)\ x = Some\ y)$
apply (*subst zip-take-length*[*symmetric*])
apply (*rule map-of-zip-is-Some*)
by (*metis length-take min-absorb2*)

lemma *map-of-zip-inj*:
$[\![distinct\ xs;\ distinct\ ys;\ length\ xs = length\ ys]\!]$
$\Longrightarrow inj\text{-}on\ (\lambda x.\ (the\ (map\text{-}of\ (zip\ xs\ ys)\ x)))\ (set\ xs)$
apply (*clarsimp simp*: *inj-on-def*)
apply (*subst* (*asm*) *map-of-zip-is-Some*, *assumption*)+

apply *clarsimp*
apply (*clarsimp simp*: *set-zip*)
by (*metis nth-eq-iff-index-eq*)

lemma *map-of-zip-inj'*:
$[\![$*distinct xs*; *distinct ys*; *length xs* $\leq$ *length ys*$]\!]$
$\Longrightarrow$ *inj-on* (λx. (*the* (*map-of* (*zip xs ys*) *x*))) (*set xs*)
apply (*subst zip-take-length*[*symmetric*])
apply (*erule map-of-zip-inj*, *simp*)
by (*metis length-take min-absorb2*)

lemma *list-all-nth*:
$[\![$*list-all P xs*; i < *length xs*$]\!]$ $\Longrightarrow$ P (*xs* ! i)
by (*metis list-all-length*)

lemma *list-all-update*:
$[\![$*list-all P xs*; i < *length xs*; $\bigwedge x$. $P\ x \Longrightarrow P\ (f\ x)$$]\!]$
$\Longrightarrow$ *list-all P* (*xs* [i := f (*xs* ! i)])
by (*metis length-list-update list-all-length nth-list-update*)

lemma *list-allI*:
$[\![$*list-all P xs*; $\bigwedge x$. $P\ x \Longrightarrow P'\ x$$]\!]$ $\Longrightarrow$ *list-all P' xs*
by (*metis list-all-length*)

lemma *list-all-imp-filter*:
list-all (λx. $f\ x \longrightarrow g\ x$) *xs* = *list-all* ($\lambda x$. $g\ x$) [$x \leftarrow$*xs* . $f\ x$]
by (*fastforce simp*: *Ball-set-list-all*[*symmetric*])

lemma *list-all-imp-filter2*:
list-all (λx. $f\ x \longrightarrow g\ x$) *xs* = *list-all* ($\lambda x$. $\neg f\ x$) [$x \leftarrow$*xs* . (λx. $\neg g\ x$) x]
by (*fastforce simp*: *Ball-set-list-all*[*symmetric*])

lemma *list-all-imp-chain*:
$[\![$*list-all* (λx. $f\ x \longrightarrow g\ x$) *xs*; *list-all* ($\lambda x$. $f'\ x \longrightarrow f\ x$) *xs*$]\!]$
$\Longrightarrow$ *list-all* (λx. $f'\ x \longrightarrow g\ x$) *xs*
by (*clarsimp simp*: *Ball-set-list-all* [*symmetric*])

lemma *inj-Pair*:
inj-on (*Pair x*) *S*
by (*rule inj-onI*, *simp*)

lemma *inj-on-split*:
inj-on f S $\Longrightarrow$ *inj-on* (λx. (z, $f\ x$)) *S*
by (*auto simp*: *inj-on-def*)

lemma *split-state-strg*:
$(\exists x.\ f\ s = x \wedge P\ x\ s) \longrightarrow P\ (f\ s)\ s$ **by** *clarsimp*

lemma *theD*:
$[\![the\ (f\ x) = y;\ \ x \in dom\ f\ ]\!] \Longrightarrow f\ x = Some\ y$
by (*auto simp add*: *dom-def*)

lemma *bspec-split*:
$[\![\ \forall (a,\ b) \in S.\ P\ a\ b;\ (a,\ b) \in S\ ]\!] \Longrightarrow P\ a\ b$
by *fastforce*

lemma *set-zip-same*:
$set\ (zip\ xs\ xs) = Id \cap (set\ xs \times set\ xs)$
by (*induct xs*) *auto*

lemma *ball-ran-updI*:
$(\forall x \in ran\ m.\ P\ x) \Longrightarrow P\ v \Longrightarrow (\forall x \in ran\ (m\ (y \mapsto v)).\ P\ x)$
by (*auto simp add*: *ran-def*)

lemma *not-psubset-eq*:
$[\![\ \neg\ A \subset B;\ A \subseteq B\ ]\!] \Longrightarrow A = B$
by *blast*

lemma *in-image-op-plus*:
$(x + y \in (+)\ x\ `\ S) = ((y :: 'a :: ring) \in S)$
by (*simp add*: *image-def*)

lemma *insert-subtract-new*:
$x \notin S \Longrightarrow (insert\ x\ S - S) = \{x\}$
by *auto*

lemma *zip-is-empty*:
$(zip\ xs\ ys = []) = (xs = [] \vee ys = [])$
by (*cases xs*; *simp*) (*cases ys*; *simp*)

lemma *minus-Suc-0-lt*:
$a \neq 0 \Longrightarrow a - Suc\ 0 < a$
by *simp*

lemma *fst-last-zip-upt*:
$zip\ [0\ ..< m]\ xs \neq [] \Longrightarrow$
$fst\ (last\ (zip\ [0\ ..< m]\ xs)) = (if\ length\ xs < m\ then\ length\ xs - 1\ else\ m - 1)$
apply (*subst last-conv-nth*, *assumption*)
apply (*simp only*: *One-nat-def*)
apply (*subst nth-zip*)
apply (*rule order-less-le-trans*[*OF minus-Suc-0-lt*])
apply (*simp add*: *zip-is-empty*)
apply *simp*

apply (*rule order-less-le-trans*[*OF minus-Suc-0-lt*])
apply (*simp add*: *zip-is-empty*)
apply *simp*
apply (*simp add*: *min-def zip-is-empty*)
done

lemma *neq-into-nprefix*:
$[\![\ x \neq take\ (length\ x)\ y\ ]\!] \Longrightarrow \neg\ x \leq y$
by (*clarsimp simp*: *prefix-def less-eq-list-def*)

lemma *suffix-eqI*:
$[\![\ suffix\ xs\ as;\ suffix\ xs\ bs;\ length\ as = length\ bs;$
$take\ (length\ as - length\ xs)\ as \leq take\ (length\ bs - length\ xs)\ bs]\!] \Longrightarrow as = bs$
by (*clarsimp elim*!: *prefixE suffixE*)

lemma *suffix-Cons-mem*:
$suffix\ (x\ \#\ xs)\ as \Longrightarrow x \in set\ as$
by (*metis in-set-conv-decomp suffix-def*)

lemma *distinct-imply-not-in-tail*:
$[\![\ distinct\ list;\ suffix\ (y\ \#\ ys)\ list]\!] \Longrightarrow y \notin set\ ys$
by (*clarsimp simp*:*suffix-def*)

lemma *list-induct-suffix* [*case-names Nil Cons*]:
assumes *nilr*: $P\ []$
and *consr*: $\bigwedge x\ xs.\ [\![P\ xs;\ suffix\ (x\ \#\ xs)\ as\ ]\!] \Longrightarrow P\ (x\ \#\ xs)$
shows *P as*
proof –
define *as'* **where** *as'* == *as*

have *suffix as as'* **unfolding** *as'-def* **by** *simp*
then show *?thesis*
proof (*induct as*)
case *Nil* **show** *?case* **by** *fact*
next
case (*Cons x xs*)

show *?case*
proof (*rule consr*)
from *Cons.prems* **show** *suffix* (*x* # *xs*) *as* **unfolding** *as'-def* **.**
then have *suffix xs as'* **by** (*auto dest*: *suffix-ConsD simp*: *as'-def*)
then show *P xs* **using** *Cons.hyps* **by** *simp*
qed
qed
qed

Parallel etc. and lemmas for list prefix

lemma *prefix-induct* [*consumes 1*, *case-names Nil Cons*]:
fixes *prefix*

assumes *np*: *prefix* $\leq$ *lst*
and *base*: $\bigwedge$*xs*. *P* [] *xs*
and *rl*: $\bigwedge$*x xs y ys*. ⟦ *x* = *y*; *xs* $\leq$ *ys*; *P xs ys* ⟧ $\Longrightarrow$ *P* (*x*#*xs*) (*y*#*ys*)
shows *P prefix lst*
using *np*
proof (*induct prefix arbitrary*: *lst*)
case *Nil* **show** *?case* **by** *fact*
next
case (*Cons x xs*)

have *prem*: (*x* # *xs*) $\leq$ *lst* **by** *fact*
then obtain *y ys* **where** *lv*: *lst* = *y* # *ys*
by (*rule prefixE*, *auto*)

have *ih*: $\bigwedge$*lst*. *xs* $\leq$ *lst* $\Longrightarrow$ *P xs lst* **by** *fact*

show *?case* **using** *prem*
by (*auto simp*: *lv intro*!: *rl ih*)
qed

lemma *not-prefix-cases*:
fixes *prefix*
assumes *pfx*: ¬ *prefix* $\leq$ *lst*
and *c1*: ⟦ *prefix* $\neq$ []; *lst* = [] ⟧ $\Longrightarrow$ *R*
and *c2*: $\bigwedge$*a as x xs*. ⟦ *prefix* = *a*#*as*; *lst* = *x*#*xs*; *x* = *a*; ¬ *as* $\leq$ *xs*⟧ $\Longrightarrow$ *R*
and *c3*: $\bigwedge$*a as x xs*. ⟦ *prefix* = *a*#*as*; *lst* = *x*#*xs*; *x* $\neq$ *a*⟧ $\Longrightarrow$ *R*
shows *R*
proof (*cases prefix*)
case *Nil* **then show** *?thesis* **using** *pfx* **by** *simp*
next
case (*Cons a as*)

have *c*: *prefix* = *a*#*as* **by** *fact*

show *?thesis*
proof (*cases lst*)
case *Nil* **then show** *?thesis*
by (*intro c1*, *simp add*: *Cons*)
next
case (*Cons x xs*)
show *?thesis*
proof (*cases x* = *a*)
case *True*
show *?thesis*
proof (*intro c2*)
show ¬ *as* $\leq$ *xs* **using** *pfx c Cons True*
by *simp*
qed *fact*+
next

case *False*
show *?thesis* **by** (*rule c3*) *fact+*
qed
qed
qed

lemma *not-prefix-induct* [*consumes 1*, *case-names Nil Neq Eq*]:
fixes *prefix*
assumes *np*: $\neg$ *prefix* $\leq$ *lst*
and *base*: $\bigwedge x\ xs.\ P\ (x\#xs)\ []$
and *r1*: $\bigwedge x\ xs\ y\ ys.\ x \neq y \Longrightarrow P\ (x\#xs)\ (y\#ys)$
and *r2*: $\bigwedge x\ xs\ y\ ys.\ [\![\ x = y;\ \neg\ xs \leq ys;\ P\ xs\ ys\ ]\!] \Longrightarrow P\ (x\#xs)\ (y\#ys)$
shows *P prefix lst*
using *np*
proof (*induct lst arbitrary*: *prefix*)
case *Nil* **then show** *?case*
by (*auto simp*: *neq-Nil-conv elim*!: *not-prefix-cases intro*!: *base*)
next
case (*Cons y ys*)

have *npfx*: $\neg$ *prefix* $\leq$ (*y* # *ys*) **by** *fact*
then obtain *x xs* **where** *pv*: *prefix* = *x* # *xs*
by (*rule not-prefix-cases*) *auto*

have *ih*: $\bigwedge$*prefix*. $\neg$ *prefix* $\leq$ *ys* $\Longrightarrow$ *P prefix ys* **by** *fact*

show *?case* **using** *npfx*
by (*simp only*: *pv*) (*erule not-prefix-cases*, *auto intro*: *r1 r2 ih*)
qed

lemma *rsubst*:
$[\![\ P\ s;\ s = t\ ]\!] \Longrightarrow P\ t$
by *simp*

lemma *ex-impE*: $((\exists x.\ P\ x) \longrightarrow Q) \Longrightarrow P\ x \Longrightarrow Q$
by *blast*

lemma *option-Some-value-independent*:
$[\![\ f\ x = Some\ v;\ \bigwedge v'.\ f\ x = Some\ v' \Longrightarrow f\ y = Some\ v'\ ]\!] \Longrightarrow f\ y = Some\ v$
by *blast*

Some int bitwise lemmas. Helpers for proofs about `NatBitwise.thy`

lemma *int-2p-eq-shiftl*:
$(2::int)\hat{\ }x = 1 << x$
by (*simp add*: *shiftl-int-def*)

lemma *nat-int-mul*:
nat (*int a* $*$ *b*) = *a* $*$ *nat b*
by (*simp add*: *nat-mult-distrib*)

lemma *int-shiftl-less-cancel*:
  $n \leq m \Longrightarrow ((x :: int) << n < y << m) = (x < y << (m - n))$
  apply (*drule le-Suc-ex*)
  apply (*clarsimp simp*: *shiftl-int-def power-add*)
  done

lemma *int-shiftl-lt-2p-bits*:
  $0 \leq (x::int) \Longrightarrow x < 1 << n \Longrightarrow \forall i \geq n.\ \neg\ x\ !!\ i$
  apply (*clarsimp simp*: *shiftl-int-def*)
  apply (*clarsimp simp*: *bin-nth-eq-mod even-iff-mod-2-eq-zero*)
  apply (*drule-tac z=2^i* **in** *less-le-trans*)
   apply *simp*
  apply *simp*
  done
— TODO: The converse should be true as well, but seems hard to prove.

lemma *int-eq-test-bit*:
  $((x :: int) = y) = (\forall i.\ test\text{-}bit\ x\ i = test\text{-}bit\ y\ i)$
  apply *simp*
  apply (*metis bin-eqI*)
  done
lemmas *int-eq-test-bitI* = *int-eq-test-bit*[*THEN iffD2*, *rule-format*]

lemma *le-nat-shrink-left*:
  $y \leq z \Longrightarrow y = Suc\ x \Longrightarrow x < z$
  by *simp*

lemma *length-ge-split*:
  $n < length\ xs \Longrightarrow \exists x\ xs'.\ xs = x\ \#\ xs' \wedge n \leq length\ xs'$
  by (*cases xs*) *auto*

end
Nondeterministic State Monad with Failure **theory** *NonDetMonad*
imports ../*Lib*
begin

State monads are used extensively in the seL4 specification. They are defined below.

13 The Monad

The basic type of the nondeterministic state monad with failure is very similar to the normal state monad. Instead of a pair consisting of result and new state, we return a set of these pairs coupled with a failure flag. Each element in the set is a potential result of the computation. The flag is *True* if there is an execution path in the computation that may have failed.

Conversely, if the flag is *False*, none of the computations resulting in the returned set can have failed.

type-synonym $('s,'a)$ *nondet-monad* $=$ $'s \Rightarrow ('a \times 's)$ *set* $\times$ *bool*

Print the type $('s,\ 'a)$ *nondet-monad* instead of its unwieldy expansion. Needs an AST translation in code, because it needs to check that the state variable $'s$ occurs twice. This comparison is not guaranteed to always work as expected (AST instances might have different decoration), but it does seem to work here.

print-ast-translation ‹

```
  let
    fun monad-tr - [t1, Ast.Appl [Ast.Constant @{type-syntax prod},
                      Ast.Appl [Ast.Constant @{type-syntax set},
                        Ast.Appl [Ast.Constant @{type-syntax prod}, t2, t3]],
                      Ast.Constant @{type-syntax bool}]] =
      if t3 = t1
      then Ast.Appl [Ast.Constant @{type-syntax nondet-monad}, t1, t2]
      else raise Match
  in [(@{type-syntax fun}, monad-tr)] end
›
```

The definition of fundamental monad functions *return* and *bind*. The monad function *return x* does not change the state, does not fail, and returns *x*.

definition
return :: $'a \Rightarrow ('s,'a)$ *nondet-monad* **where**
return a $\equiv \lambda s.\ (\{(a,s)\},$*False*$)$

The monad function *bind f g*, also written $f >>= g$, is the execution of *f* followed by the execution of *g*. The function *g* takes the result value *and* the result state of *f* as parameter. The definition says that the result of the combined operation is the union of the set of sets that is created by *g* applied to the result sets of *f*. The combined operation may have failed, if *f* may have failed or *g* may have failed on any of the results of *f*.

definition
bind :: $('s,\ 'a)$ *nondet-monad* $\Rightarrow ('a \Rightarrow ('s,\ 'b)$ *nondet-monad*$) \Rightarrow$
$('s,\ 'b)$ *nondet-monad* (**infixl** $>>=$ *60*)
where
bind f g $\equiv \lambda s.\ (\bigcup (fst\ `\ case\text{-}prod\ g\ `\ fst\ (f\ s)),$
$True \in snd\ `\ case\text{-}prod\ g\ `\ fst\ (f\ s) \vee snd\ (f\ s))$

Sometimes it is convenient to write *bind* in reverse order.

abbreviation(*input*)
bind-rev :: $('c \Rightarrow ('a,\ 'b)$ *nondet-monad*$) \Rightarrow ('a,\ 'c)$ *nondet-monad* $\Rightarrow$
$('a,\ 'b)$ *nondet-monad* (**infixl** $=<<$ *60*) **where**
$g =<< f \equiv f >>= g$

The basic accessor functions of the state monad. *get* returns the current state as result, does not fail, and does not change the state. *put s* returns nothing (*unit*), changes the current state to *s* and does not fail.

definition
  $get :: ('s,'s)\ nondet\text{-}monad$ **where**
  $get \equiv \lambda s.\ (\{(s,s)\},\ False)$

definition
  $put :: 's \Rightarrow ('s,\ unit)\ nondet\text{-}monad$ **where**
  $put\ s \equiv \lambda\text{-}.\ (\{((),s)\},\ False)$

13.1 Nondeterminism

Basic nondeterministic functions. *select A* chooses an element of the set *A*, does not change the state, and does not fail (even if the set is empty). *f OR g* executes *f* or executes *g*. It retuns the union of results of *f* and *g*, and may have failed if either may have failed.

definition
  $select :: 'a\ set \Rightarrow ('s,'a)\ nondet\text{-}monad$ **where**
  $select\ A \equiv \lambda s.\ (A \times \{s\},\ False)$

definition
  $alternative :: ('s,'a)\ nondet\text{-}monad \Rightarrow ('s,'a)\ nondet\text{-}monad \Rightarrow ('s,'a)\ nondet\text{-}monad$
  (**infixl** *OR 20*)
where
  $f\ OR\ g \equiv \lambda s.\ (fst\ (f\ s) \cup fst\ (g\ s),\ snd\ (f\ s) \vee snd\ (g\ s))$

Alternative notation for *OR*

notation (*xsymbols*) *alternative* (**infixl** $\sqcap$ *20*)

A variant of *select* that takes a pair. The first component is a set as in normal *select*, the second component indicates whether the execution failed. This is useful to lift monads between different state spaces.

definition
  $select\text{-}f :: 'a\ set \times bool \Rightarrow ('s,'a)\ nondet\text{-}monad$ **where**
  $select\text{-}f\ S \equiv \lambda s.\ (fst\ S \times \{s\},\ snd\ S)$

select-state takes a relationship between states, and outputs nondeterministically a state related to the input state.

definition
  $state\text{-}select :: ('s \times 's)\ set \Rightarrow ('s,\ unit)\ nondet\text{-}monad$
where
  $state\text{-}select\ r \equiv \lambda s.\ ((\lambda x.\ ((),\ x))\ `\ \{s'.\ (s,\ s') \in r\},\ \neg\ (\exists s'.\ (s,\ s') \in r))$

13.2 Failure

The monad function that always fails. Returns an empty set of results and sets the failure flag.

definition
  fail :: $('s, 'a)$ *nondet-monad* **where**
 $fail \equiv \lambda s.\ (\{\},\ True)$

Assertions: fail if the property P is not true

definition
  assert :: *bool* $\Rightarrow$ $('a,\ unit)$ *nondet-monad* **where**
 $assert\ P \equiv if\ P\ then\ return\ ()\ else\ fail$

Fail if the value is *None*, return result v for *Some* v

definition
  assert-opt :: $'a\ option \Rightarrow ('b,\ 'a)$ *nondet-monad* **where**
 $assert\text{-}opt\ v \equiv case\ v\ of\ None \Rightarrow fail \mid Some\ v \Rightarrow return\ v$

An assertion that also can introspect the current state.

definition
  state-assert :: $('s \Rightarrow bool) \Rightarrow ('s,\ unit)$ *nondet-monad*
where
  $state\text{-}assert\ P \equiv get >>= (\lambda s.\ assert\ (P\ s))$

13.3 Generic functions on top of the state monad

Apply a function to the current state and return the result without changing the state.

definition
  gets :: $('s \Rightarrow 'a) \Rightarrow ('s,\ 'a)$ *nondet-monad* **where**
 $gets\ f \equiv get >>= (\lambda s.\ return\ (f\ s))$

Modify the current state using the function passed in.

definition
  modify :: $('s \Rightarrow 's) \Rightarrow ('s,\ unit)$ *nondet-monad* **where**
 $modify\ f \equiv get >>= (\lambda s.\ put\ (f\ s))$

lemma *simpler-gets-def*: $gets\ f = (\lambda s.\ (\{(f\ s,\ s)\},\ False))$
  apply (*simp add*: *gets-def return-def bind-def get-def*)
  done

lemma *simpler-modify-def*:
  $modify\ f = (\lambda s.\ (\{((),\ f\ s)\},\ False))$
  by (*simp add*: *modify-def bind-def get-def put-def*)

Execute the given monad when the condition is true, return () otherwise.

definition

$when :: bool \Rightarrow ('s,\ unit)\ nondet\text{-}monad \Rightarrow$
$('s,\ unit)\ nondet\text{-}monad$ **where**
$when\ P\ m \equiv if\ P\ then\ m\ else\ return\ ()$

Execute the given monad unless the condition is true, return () otherwise.

definition
$unless :: bool \Rightarrow ('s,\ unit)\ nondet\text{-}monad \Rightarrow$
$('s,\ unit)\ nondet\text{-}monad$ **where**
$unless\ P\ m \equiv when\ (\neg P)\ m$

Perform a test on the current state, performing the left monad if the result is true or the right monad if the result is false.

definition
$condition :: ('s \Rightarrow bool) \Rightarrow ('s,\ 'r)\ nondet\text{-}monad \Rightarrow ('s,\ 'r)\ nondet\text{-}monad \Rightarrow ('s,\ 'r)\ nondet\text{-}monad$
where
$condition\ P\ L\ R \equiv \lambda s.\ if\ (P\ s)\ then\ (L\ s)\ else\ (R\ s)$

notation (**output**)
$condition\ \ ((condition\ (\text{-})//\ \ (\text{-})//\ \ (\text{-}))\ [1000,1000,1000]\ 1000)$

Apply an option valued function to the current state, fail if it returns *None*, return *v* if it returns *Some v*.

definition
$gets\text{-}the :: ('s \Rightarrow 'a\ option) \Rightarrow ('s,\ 'a)\ nondet\text{-}monad$ **where**
$gets\text{-}the\ f \equiv gets\ f\ >>=\ assert\text{-}opt$

Get a map (such as a heap) from the current state and apply an argument to the map. Fail if the map returns *None*, otherwise return the value.

definition
$gets\text{-}map :: ('s \Rightarrow 'a \Rightarrow 'b\ option) \Rightarrow 'a \Rightarrow ('s,\ 'b)\ nondet\text{-}monad$ **where**
$gets\text{-}map\ f\ p \equiv gets\ f\ >>=\ (\lambda m.\ assert\text{-}opt\ (m\ p))$

13.4 The Monad Laws

A more expanded definition of *bind*

lemma *bind-def'*:
$(f\ >>=\ g) \equiv$
$\lambda s.\ (\{(r'',\ s'').\ \exists (r',\ s') \in fst\ (f\ s).\ (r'',\ s'') \in fst\ (g\ r'\ s')\ \},$
$snd\ (f\ s) \vee (\exists (r',\ s') \in fst\ (f\ s).\ snd\ (g\ r'\ s')))$
apply (*rule eq-reflection*)
apply (*auto simp add*: *bind-def split-def Let-def*)
done

Each monad satisfies at least the following three laws.

return is absorbed at the left of a (>>=), applying the return value directly:

lemma *return-bind* [*simp*]: (*return* x >>= f) = f x
  by (*simp add*: *return-def bind-def*)

return is absorbed on the right of a (>>=)

lemma *bind-return* [*simp*]: (m >>= *return*) = m
  apply (*rule ext*)
  apply (*simp add*: *bind-def return-def split-def*)
  done

(>>=) is associative

lemma *bind-assoc*:
  fixes m :: ('a,'b) *nondet-monad*
  fixes f :: 'b ⇒ ('a,'c) *nondet-monad*
  fixes g :: 'c ⇒ ('a,'d) *nondet-monad*
  shows (m >>= f) >>= g = m >>= (λx. f x >>= g)
  apply (*unfold bind-def Let-def split-def*)
  apply (*rule ext*)
  apply *clarsimp*
  apply (*auto intro*: *rev-image-eqI*)
  done

14 Adding Exceptions

The type ('s, 'a) *nondet-monad* gives us nondeterminism and failure. We now extend this monad with exceptional return values that abort normal execution, but can be handled explicitly. We use the sum type to indicate exceptions.

In ('s, 'e + 'a) *nondet-monad*, 's is the state, 'e is an exception, and 'a is a normal return value.

This new type itself forms a monad again. Since type classes in Isabelle are not powerful enough to express the class of monads, we provide new names for the *return* and (>>=) functions in this monad. We call them *returnOk* (for normal return values) and *bindE* (for composition). We also define *throwError* to return an exceptional value.

definition
  returnOk :: 'a ⇒ ('s, 'e + 'a) *nondet-monad* **where**
  returnOk ≡ *return o Inr*

definition
  throwError :: 'e ⇒ ('s, 'e + 'a) *nondet-monad* **where**
  throwError ≡ *return o Inl*

Lifting a function over the exception type: if the input is an exception, return that exception; otherwise continue execution.

definition

$lift :: ('a \Rightarrow ('s, 'e + 'b)\ nondet\text{-}monad) \Rightarrow$
$\quad 'e + 'a \Rightarrow ('s, 'e + 'b)\ nondet\text{-}monad$

where

$lift\ f\ v \equiv case\ v\ of\ Inl\ e \Rightarrow throwError\ e$
$\quad \mid Inr\ v' \Rightarrow f\ v'$

The definition of (>>=) in the exception monad (new name *bindE*): the same as normal (>>=), but the right-hand side is skipped if the left-hand side produced an exception.

definition

$bindE :: ('s, 'e + 'a)\ nondet\text{-}monad \Rightarrow$
$\quad ('a \Rightarrow ('s, 'e + 'b)\ nondet\text{-}monad) \Rightarrow$
$\quad ('s, 'e + 'b)\ nondet\text{-}monad$ (**infixl** $>>=E\ 60$)

where

$bindE\ f\ g \equiv bind\ f\ (lift\ g)$

Lifting a normal nondeterministic monad into the exception monad is achieved by always returning its result as normal result and never throwing an exception.

definition

$liftE :: ('s,'a)\ nondet\text{-}monad \Rightarrow ('s, 'e+'a)\ nondet\text{-}monad$

where

$liftE\ f \equiv f >>= (\lambda r.\ return\ (Inr\ r))$

Since the underlying type and *return* function changed, we need new definitions for when and unless:

definition

$whenE :: bool \Rightarrow ('s, 'e + unit)\ nondet\text{-}monad \Rightarrow$
$\quad ('s, 'e + unit)\ nondet\text{-}monad$

where

$whenE\ P\ f \equiv if\ P\ then\ f\ else\ returnOk\ ()$

definition

$unlessE :: bool \Rightarrow ('s, 'e + unit)\ nondet\text{-}monad \Rightarrow$
$\quad ('s, 'e + unit)\ nondet\text{-}monad$

where

$unlessE\ P\ f \equiv if\ P\ then\ returnOk\ ()\ else\ f$

Throwing an exception when the parameter is *None*, otherwise returning v for *Some v*.

definition

$throw\text{-}opt :: 'e \Rightarrow 'a\ option \Rightarrow ('s, 'e + 'a)\ nondet\text{-}monad$ **where**
$throw\text{-}opt\ ex\ x \equiv$
$case\ x\ of\ None \Rightarrow throwError\ ex \mid Some\ v \Rightarrow returnOk\ v$

Failure in the exception monad is redefined in the same way as *whenE* and *unlessE*, with *returnOk* instead of *return*.

definition
  assertE :: *bool* ⇒ ('*a*, '*e* + *unit*) *nondet-monad* **where**
 assertE P ≡ *if P then returnOk* () *else fail*

14.1 Monad Laws for the Exception Monad

More direct definition of *liftE*:

lemma *liftE-def2*:
  liftE f = (λ*s*. ((λ(*v*,*s*'). (*Inr v*, *s*')) ' *fst* (*f s*), *snd* (*f s*)))
  by (*auto simp*: *liftE-def return-def split-def bind-def*)

Left *returnOk* absorbtion over (>>=*E*):

lemma *returnOk-bindE* [*simp*]: (*returnOk x* >>=*E f*) = *f x*
  apply (*unfold bindE-def returnOk-def*)
  apply (*clarsimp simp*: *lift-def*)
  done

lemma *lift-return* [*simp*]:
  lift (*return* ∘ *Inr*) = *return*
  by (*rule ext*)
    (*simp add*: *lift-def throwError-def split*: *sum.splits*)

Right *returnOk* absorbtion over (>>=*E*):

lemma *bindE-returnOk* [*simp*]: (*m* >>=*E returnOk*) = *m*
  by (*simp add*: *bindE-def returnOk-def*)

Associativity of (>>=*E*):

lemma *bindE-assoc*:
  (*m* >>=*E f*) >>=*E g* = *m* >>=*E* (λ*x*. *f x* >>=*E g*)
  apply (*simp add*: *bindE-def bind-assoc*)
  apply (*rule arg-cong* [**where** *f*=λ*x*. *m* >>= *x*])
  apply (*rule ext*)
  apply (*case-tac x*, *simp-all add*: *lift-def throwError-def*)
  done

returnOk could also be defined via *liftE*:

lemma *returnOk-liftE*:
  returnOk x = *liftE* (*return x*)
  by (*simp add*: *liftE-def returnOk-def*)

Execution after throwing an exception is skipped:

lemma *throwError-bindE* [*simp*]:
  (*throwError E* >>=*E f*) = *throwError E*
  by (*simp add*: *bindE-def bind-def throwError-def lift-def return-def*)

15 Syntax

This section defines traditional Haskell-like do-syntax for the state monad in Isabelle.

15.1 Syntax for the Nondeterministic State Monad

We use *K-bind* to syntactically indicate the case where the return argument of the left side of a (>>=) is ignored

definition
  K-bind-def [*iff*]: *K-bind* ≡ λ*x y*. *x*

nonterminal
  dobinds **and** *dobind* **and** *nobind*

syntax
```
  -dobind    :: [pttrn, 'a] => dobind            ((- <−/ -) 10)
             :: dobind => dobinds                (-)
  -nobind    :: 'a => dobind                     (-)
  -dobinds   :: [dobind, dobinds] => dobinds     ((-);//(-))

  -do        :: [dobinds, 'a] => 'a              ((do ((-);//(-))//od) 100)
```
syntax (*xsymbols*)
```
  -dobind    :: [pttrn, 'a] => dobind            ((- ←/ -) 10)
```

translations
```
  -do (-dobinds b bs) e  == -do b (-do bs e)
  -do (-nobind b) e      == b >>= (CONST K-bind e)
  do x <− a; e od        == a >>= (λx. e)
```

Syntax examples:

lemma *do x* ← *return 1*;
        return (*2*::*nat*);
        return x
      od =
      return 1 >>=
      (λ*x*. *return* (*2*::*nat*) >>=
          K-bind (*return x*))
  by (*rule refl*)

lemma *do x* ← *return 1*;
        return 2;
        return x
      od = *return 1*
  by *simp*

15.2 Syntax for the Exception Monad

Since the exception monad is a different type, we need to syntactically distinguish it in the syntax. We use *doE*/*odE* for this, but can re-use most of the productions from *do*/*od* above.

syntax
  -doE :: [*dobinds*, *'a*] => *'a* ((*doE* ((-);//(-))//*odE*) *100*)

translations
  -doE (*-dobinds b bs*) *e* == *-doE b* (*-doE bs e*)
  -doE (*-nobind b*) *e* == *b* >>=*E* (*CONST K-bind e*)
  doE x <− *a*; *e odE* == *a* >>=*E* (λ*x*. *e*)

Syntax examples:

lemma *doE x* ← *returnOk 1*;
    returnOk (*2*::*nat*);
    returnOk x
  odE =
  returnOk 1 >>=*E*
  (λ*x*. *returnOk* (*2*::*nat*) >>=*E*
    K-bind (*returnOk x*))
  by (*rule refl*)

lemma *doE x* ← *returnOk 1*;
    returnOk 2;
    returnOk x
  odE = *returnOk 1*
  by *simp*

16 Library of Monadic Functions and Combinators

Lifting a normal function into the monad type:

definition
  liftM :: (*'a* ⇒ *'b*) ⇒ (*'s*,*'a*) *nondet-monad* ⇒ (*'s*, *'b*) *nondet-monad*
where
  liftM f m ≡ *do x* ← *m*; *return* (*f x*) *od*

The same for the exception monad:

definition
  liftME :: (*'a* ⇒ *'b*) ⇒ (*'s*,*'e*+*'a*) *nondet-monad* ⇒ (*'s*,*'e*+*'b*) *nondet-monad*
where
  liftME f m ≡ *doE x* ← *m*; *returnOk* (*f x*) *odE*

Run a sequence of monads from left to right, ignoring return values.

definition

$sequence\text{-}x :: ('s, 'a)\ nondet\text{-}monad\ list \Rightarrow ('s, unit)\ nondet\text{-}monad$
where
$sequence\text{-}x\ xs \equiv foldr\ (\lambda x\ y.\ x >>= (\lambda\text{-}.\ y))\ xs\ (return\ ())$

Map a monadic function over a list by applying it to each element of the list from left to right, ignoring return values.

definition
$mapM\text{-}x :: ('a \Rightarrow ('s,'b)\ nondet\text{-}monad) \Rightarrow 'a\ list \Rightarrow ('s, unit)\ nondet\text{-}monad$
where
$mapM\text{-}x\ f\ xs \equiv sequence\text{-}x\ (map\ f\ xs)$

Map a monadic function with two parameters over two lists, going through both lists simultaneously, left to right, ignoring return values.

definition
$zipWithM\text{-}x :: ('a \Rightarrow 'b \Rightarrow ('s,'c)\ nondet\text{-}monad) \Rightarrow$
$\quad 'a\ list \Rightarrow 'b\ list \Rightarrow ('s, unit)\ nondet\text{-}monad$
where
$zipWithM\text{-}x\ f\ xs\ ys \equiv sequence\text{-}x\ (zipWith\ f\ xs\ ys)$

The same three functions as above, but returning a list of return values instead of *unit*

definition
$sequence :: ('s, 'a)\ nondet\text{-}monad\ list \Rightarrow ('s, 'a\ list)\ nondet\text{-}monad$
where
$sequence\ xs \equiv let\ mcons = (\lambda p\ q.\ p >>= (\lambda x.\ q >>= (\lambda y.\ return\ (x\#y))))$
$\quad in\ foldr\ mcons\ xs\ (return\ [])$

definition
$mapM :: ('a \Rightarrow ('s,'b)\ nondet\text{-}monad) \Rightarrow 'a\ list \Rightarrow ('s, 'b\ list)\ nondet\text{-}monad$
where
$mapM\ f\ xs \equiv sequence\ (map\ f\ xs)$

definition
$zipWithM :: ('a \Rightarrow 'b \Rightarrow ('s,'c)\ nondet\text{-}monad) \Rightarrow$
$\quad 'a\ list \Rightarrow 'b\ list \Rightarrow ('s, 'c\ list)\ nondet\text{-}monad$
where
$zipWithM\ f\ xs\ ys \equiv sequence\ (zipWith\ f\ xs\ ys)$

definition
$foldM :: ('b \Rightarrow 'a \Rightarrow ('s, 'a)\ nondet\text{-}monad) \Rightarrow 'b\ list \Rightarrow 'a \Rightarrow ('s, 'a)\ nondet\text{-}monad$
where
$foldM\ m\ xs\ a \equiv foldr\ (\lambda p\ q.\ q >>= m\ p)\ xs\ (return\ a)$

definition
$foldME ::('b \Rightarrow 'a \Rightarrow ('s,('e + 'b))\ nondet\text{-}monad) \Rightarrow 'b \Rightarrow 'a\ list \Rightarrow ('s, ('e + 'b))\ nondet\text{-}monad$
where $foldME\ m\ a\ xs \equiv foldr\ (\lambda p\ q.\ q >>=E\ swp\ m\ p)\ xs\ (returnOk\ a)$

The sequence and map functions above for the exception monad, with and without lists of return value

definition
sequenceE-x :: ('s, 'e+'a) nondet-monad list ⇒ ('s, 'e+unit) nondet-monad
where
sequenceE-x xs ≡ foldr (λx y. doE - <− x; y odE) xs (returnOk ())

definition
mapME-x :: ('a ⇒ ('s,'e+'b) nondet-monad) ⇒ 'a list ⇒
('s,'e+unit) nondet-monad
where
mapME-x f xs ≡ sequenceE-x (map f xs)

definition
sequenceE :: ('s, 'e+'a) nondet-monad list ⇒ ('s, 'e+'a list) nondet-monad
where
sequenceE xs ≡ let mcons = (λp q. p >>=E (λx. q >>=E (λy. returnOk (x#y))))
in foldr mcons xs (returnOk [])

definition
mapME :: ('a ⇒ ('s,'e+'b) nondet-monad) ⇒ 'a list ⇒
('s,'e+'b list) nondet-monad
where
mapME f xs ≡ sequenceE (map f xs)

Filtering a list using a monadic function as predicate:

primrec
filterM :: ('a ⇒ ('s, bool) nondet-monad) ⇒ 'a list ⇒ ('s, 'a list) nondet-monad
where
filterM P [] = return []
| *filterM P (x # xs)* = do
b <− P x;
ys <− filterM P xs;
return (if b then (x # ys) else ys)
od

17 Catching and Handling Exceptions

Turning an exception monad into a normal state monad by catching and handling any potential exceptions:

definition
catch :: ('s, 'e + 'a) nondet-monad ⇒
('e ⇒ ('s, 'a) nondet-monad) ⇒
('s, 'a) nondet-monad (**infix** <catch> 10)
where
f <catch> handler ≡

```
do x ← f;
   case x of
     Inr b ⇒ return b
   | Inl e ⇒ handler e
od
```

Handling exceptions, but staying in the exception monad. The handler may throw a type of exceptions different from the left side.

definition
```
  handleE' :: ('s, 'e1 + 'a) nondet-monad ⇒
              ('e1 ⇒ ('s, 'e2 + 'a) nondet-monad) ⇒
              ('s, 'e2 + 'a) nondet-monad (infix <handle2> 10)
```
where
```
  f <handle2> handler ≡
   do
      v ← f;
      case v of
        Inl e ⇒ handler e
      | Inr v' ⇒ return (Inr v')
   od
```

A type restriction of the above that is used more commonly in practice: the exception handle (potentially) throws exception of the same type as the left-hand side.

definition
```
  handleE :: ('s, 'x + 'a) nondet-monad ⇒
             ('x ⇒ ('s, 'x + 'a) nondet-monad) ⇒
             ('s, 'x + 'a) nondet-monad (infix <handle> 10)
```
where
```
  handleE ≡ handleE'
```

Handling exceptions, and additionally providing a continuation if the left-hand side throws no exception:

definition
```
  handle-elseE :: ('s, 'e + 'a) nondet-monad ⇒
                  ('e ⇒ ('s, 'ee + 'b) nondet-monad) ⇒
                  ('a ⇒ ('s, 'ee + 'b) nondet-monad) ⇒
                  ('s, 'ee + 'b) nondet-monad
  (- <handle> - <else> - 10)
```
where
```
  f <handle> handler <else> continue ≡
   do v ← f;
   case v of Inl e  ⇒ handler e
           | Inr v' ⇒ continue v'
   od
```

17.1 Loops

Loops are handled using the following inductive predicate; non-termination is represented using the failure flag of the monad.

inductive-set
  whileLoop-results :: $('r \Rightarrow 's \Rightarrow bool) \Rightarrow ('r \Rightarrow ('s, 'r)\ nondet\text{-}monad) \Rightarrow ((('r \times 's)\ option) \times (('r \times 's)\ option))\ set$
  for $C\ B$
where
  $[\![\ \neg\ C\ r\ s\ ]\!] \Longrightarrow (Some\ (r,\ s),\ Some\ (r,\ s)) \in whileLoop\text{-}results\ C\ B$
  $|\ [\![\ C\ r\ s;\ snd\ (B\ r\ s)\ ]\!] \Longrightarrow (Some\ (r,\ s),\ None) \in whileLoop\text{-}results\ C\ B$
  $|\ [\![\ C\ r\ s;\ (r',\ s') \in fst\ (B\ r\ s);\ (Some\ (r',\ s'),\ z) \in whileLoop\text{-}results\ C\ B\ \ ]\!]$
    $\Longrightarrow (Some\ (r,\ s),\ z) \in whileLoop\text{-}results\ C\ B$

inductive-cases *whileLoop-results-cases-valid*: $(Some\ x,\ Some\ y) \in whileLoop\text{-}results\ C\ B$
inductive-cases *whileLoop-results-cases-fail*: $(Some\ x,\ None) \in whileLoop\text{-}results\ C\ B$
inductive-simps *whileLoop-results-simps*: $(Some\ x,\ y) \in whileLoop\text{-}results\ C\ B$
inductive-simps *whileLoop-results-simps-valid*: $(Some\ x,\ Some\ y) \in whileLoop\text{-}results\ C\ B$
inductive-simps *whileLoop-results-simps-start-fail* [*simp*]: $(None,\ x) \in whileLoop\text{-}results\ C\ B$

inductive
  whileLoop-terminates :: $('r \Rightarrow 's \Rightarrow bool) \Rightarrow ('r \Rightarrow ('s, 'r)\ nondet\text{-}monad) \Rightarrow 'r \Rightarrow 's \Rightarrow bool$
  for $C\ B$
where
  $\neg\ C\ r\ s \Longrightarrow whileLoop\text{-}terminates\ C\ B\ r\ s$
  $|\ [\![\ C\ r\ s;\ \forall (r',\ s') \in fst\ (B\ r\ s).\ whileLoop\text{-}terminates\ C\ B\ r'\ s'\ ]\!]$
    $\Longrightarrow whileLoop\text{-}terminates\ C\ B\ r\ s$

inductive-cases *whileLoop-terminates-cases*: $whileLoop\text{-}terminates\ C\ B\ r\ s$
inductive-simps *whileLoop-terminates-simps*: $whileLoop\text{-}terminates\ C\ B\ r\ s$

definition
  $whileLoop\ C\ B \equiv (\lambda r\ s.$
   $(\{(r',s').\ (Some\ (r,\ s),\ Some\ (r',\ s')) \in whileLoop\text{-}results\ C\ B\},$
    $(Some\ (r,\ s),\ None) \in whileLoop\text{-}results\ C\ B \lor (\neg\ whileLoop\text{-}terminates\ C\ B\ r\ s)))$

notation (**output**)
  whileLoop ((*whileLoop* (-)// (-)) [*1000*, *1000*] *1000*)

definition
  whileLoopE :: $('r \Rightarrow 's \Rightarrow bool) \Rightarrow ('r \Rightarrow ('s, 'e + 'r)\ nondet\text{-}monad)$
    $\Rightarrow 'r \Rightarrow 's \Rightarrow (('e + 'r) \times 's)\ set \times bool$
where

whileLoopE C body $\equiv$
$\lambda r.$ *whileLoop* $(\lambda r\ s.\ (\mathit{case}\ r\ \mathit{of}\ \mathit{Inr}\ v \Rightarrow C\ v\ s \mid \text{-} \Rightarrow \mathit{False}))\ (\mathit{lift\ body})\ (\mathit{Inr}\ r)$

notation (**output**)
whileLoopE ((*whileLoopE* (-)// (-)) [*1000*, *1000*] *1000*)

18 Hoare Logic

18.1 Validity

This section defines a Hoare logic for partial correctness for the nondeterministic state monad as well as the exception monad. The logic talks only about the behaviour part of the monad and ignores the failure flag.

The logic is defined semantically. Rules work directly on the validity predicate.

In the nondeterministic state monad, validity is a triple of precondition, monad, and postcondition. The precondition is a function from state to bool (a state predicate), the postcondition is a function from return value to state to bool. A triple is valid if for all states that satisfy the precondition, all result values and result states that are returned by the monad satisfy the postcondition. Note that if the computation returns the empty set, the triple is trivially valid. This means *assert* P does not require us to prove that P holds, but rather allows us to assume P! Proving non-failure is done via separate predicate and calculus (see below).

definition
valid :: $('s \Rightarrow \mathit{bool}) \Rightarrow ('s,'a)\ \mathit{nondet\text{-}monad} \Rightarrow ('a \Rightarrow 's \Rightarrow \mathit{bool}) \Rightarrow \mathit{bool}$
(⦃-⦄/ - /⦃-⦄)
where
$⦃P⦄\ f\ ⦃Q⦄ \equiv \forall s.\ P\ s \longrightarrow (\forall (r,s') \in \mathit{fst}\ (f\ s).\ Q\ r\ s')$

We often reason about invariant predicates. The following provides shorthand syntax that avoids repeating potentially long predicates.

abbreviation (*input*)
invariant :: $('s,'a)\ \mathit{nondet\text{-}monad} \Rightarrow ('s \Rightarrow \mathit{bool}) \Rightarrow \mathit{bool}$ (- ⦃-⦄ [*59*,*0*] *60*)
where
invariant f P $\equiv ⦃P⦄\ f\ ⦃\lambda\text{-}.\ P⦄$

Validity for the exception monad is similar and build on the standard validity above. Instead of one postcondition, we have two: one for normal and one for exceptional results.

definition
validE :: $('s \Rightarrow \mathit{bool}) \Rightarrow ('s,\ 'a + 'b)\ \mathit{nondet\text{-}monad} \Rightarrow$
$('b \Rightarrow 's \Rightarrow \mathit{bool}) \Rightarrow$
$('a \Rightarrow 's \Rightarrow \mathit{bool}) \Rightarrow \mathit{bool}$
(⦃-⦄/ - /(⦃-⦄,/ ⦃-⦄))

where
$\{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\} \equiv \{\!|P|\!\}\ f\ \{\!|\ \lambda v\ s.\ \mathit{case}\ v\ \mathit{of}\ \mathit{Inr}\ r \Rightarrow Q\ r\ s \mid \mathit{Inl}\ e \Rightarrow E\ e\ s\ |\!\}$

The following two instantiations are convenient to separate reasoning for exceptional and normal case.

definition
$\mathit{validE\text{-}R} :: ('s \Rightarrow \mathit{bool}) \Rightarrow ('s, 'e + 'a)\ \mathit{nondet\text{-}monad} \Rightarrow$
$('a \Rightarrow 's \Rightarrow \mathit{bool}) \Rightarrow \mathit{bool}$
$(\{\!|\text{-}|\!\}/\ \text{-}\ /\{\!|\text{-}|\!\},\ -)$
where
$\{\!|P|\!\}\ f\ \{\!|Q|\!\},- \equiv \mathit{validE}\ P\ f\ Q\ (\lambda x\ y.\ \mathit{True})$

definition
$\mathit{validE\text{-}E} :: ('s \Rightarrow \mathit{bool}) \Rightarrow ('s, 'e + 'a)\ \mathit{nondet\text{-}monad} \Rightarrow$
$('e \Rightarrow 's \Rightarrow \mathit{bool}) \Rightarrow \mathit{bool}$
$(\{\!|\text{-}|\!\}/\ \text{-}\ /-,\ \{\!|\text{-}|\!\})$
where
$\{\!|P|\!\}\ f\ -,\{\!|Q|\!\} \equiv \mathit{validE}\ P\ f\ (\lambda x\ y.\ \mathit{True})\ Q$

Abbreviations for trivial preconditions:

abbreviation(*input*)
$\mathit{top} :: 'a \Rightarrow \mathit{bool}\ (\top)$
where
$\top \equiv \lambda\text{-}.\ \mathit{True}$

abbreviation(*input*)
$\mathit{bottom} :: 'a \Rightarrow \mathit{bool}\ (\bot)$
where
$\bot \equiv \lambda\text{-}.\ \mathit{False}$

Abbreviations for trivial postconditions (taking two arguments):

abbreviation(*input*)
$\mathit{toptop} :: 'a \Rightarrow 'b \Rightarrow \mathit{bool}\ (\top\top)$
where
$\top\top \equiv \lambda\text{-}\ \text{-}.\ \mathit{True}$

abbreviation(*input*)
$\mathit{botbot} :: 'a \Rightarrow 'b \Rightarrow \mathit{bool}\ (\bot\bot)$
where
$\bot\bot \equiv \lambda\text{-}\ \text{-}.\ \mathit{False}$

Lifting $\wedge$ and $\vee$ over two arguments. Lifting $\wedge$ and $\vee$ over one argument is already defined (written *and* and *or*).

definition
$\mathit{bipred\text{-}conj} :: ('a \Rightarrow 'b \Rightarrow \mathit{bool}) \Rightarrow ('a \Rightarrow 'b \Rightarrow \mathit{bool}) \Rightarrow ('a \Rightarrow 'b \Rightarrow \mathit{bool})$
(**infixl** *And 96*)
where
$\mathit{bipred\text{-}conj}\ P\ Q \equiv \lambda x\ y.\ P\ x\ y \wedge Q\ x\ y$

definition
bipred-disj :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool)$
(**infixl** *Or 91*)
where
$bipred\text{-}disj\ P\ Q \equiv \lambda x\ y.\ P\ x\ y \lor Q\ x\ y$

18.2 Determinism

A monad of type *nondet-monad* is deterministic iff it returns exactly one state and result and does not fail

definition
det :: $('a,'s)$ *nondet-monad* $\Rightarrow$ *bool*
where
$det\ f \equiv \forall s.\ \exists r.\ f\ s = (\{r\},False)$

A deterministic *nondet-monad* can be turned into a normal state monad:

definition
the-run-state :: $('s,'a)$ *nondet-monad* $\Rightarrow 's \Rightarrow 'a \times 's$
where
$the\text{-}run\text{-}state\ M \equiv \lambda s.\ THE\ s'.\ fst\ (M\ s) = \{s'\}$

18.3 Non-Failure

With the failure flag, we can formulate non-failure separately from validity. A monad m does not fail under precondition P, if for no start state in that precondition it sets the failure flag.

definition
no-fail :: $('s \Rightarrow bool) \Rightarrow ('s,'a)$ *nondet-monad* $\Rightarrow$ *bool*
where
$no\text{-}fail\ P\ m \equiv \forall s.\ P\ s \longrightarrow \neg\ (snd\ (m\ s))$

It is often desired to prove non-failure and a Hoare triple simultaneously, as the reasoning is often similar. The following definitions allow such reasoning to take place.

definition
validNF ::$('s \Rightarrow bool) \Rightarrow ('s,'a)$ *nondet-monad* $\Rightarrow ('a \Rightarrow 's \Rightarrow bool) \Rightarrow bool$
({|-|}/ - /{|-|}!)
where
$validNF\ P\ f\ Q \equiv valid\ P\ f\ Q \land no\text{-}fail\ P\ f$

definition
validE-NF :: $('s \Rightarrow bool) \Rightarrow ('s,\ 'a + 'b)$ *nondet-monad* $\Rightarrow$
$('b \Rightarrow 's \Rightarrow bool) \Rightarrow$
$('a \Rightarrow 's \Rightarrow bool) \Rightarrow bool$
({|-|}/ - /({|-|},/ {|-|}!))
where

$validE\text{-}NF\ P\ f\ Q\ E \equiv validE\ P\ f\ Q\ E \wedge no\text{-}fail\ P\ f$

lemma *validE-NF-alt-def*:
$\{\!|\ P\ |\!\}\ B\ \{\!|\ Q\ |\!\},\{\!|\ E\ |\!\}! = \{\!|\ P\ |\!\}\ B\ \{\!|\ \lambda v\ s.\ case\ v\ of\ Inl\ e \Rightarrow E\ e\ s \mid Inr\ r \Rightarrow Q\ r\ s\ |\!\}!$
by (*clarsimp simp*: *validE-NF-def validE-def validNF-def*)

Usually, well-formed monads constructed from the primitives above will have the following property: if they return an empty set of results, they will have the failure flag set.

definition
$empty\text{-}fail :: ('s,'a)\ nondet\text{-}monad \Rightarrow bool$
where
$empty\text{-}fail\ m \equiv \forall s.\ fst\ (m\ s) = \{\} \longrightarrow snd\ (m\ s)$

Useful in forcing otherwise unknown executions to have the *empty-fail* property.

definition
$mk\text{-}ef :: 'a\ set \times bool \Rightarrow 'a\ set \times bool$
where
$mk\text{-}ef\ S \equiv (fst\ S,\ fst\ S = \{\} \vee snd\ S)$

19 Basic exception reasoning

The following predicates *no-throw* and *no-return* allow reasoning that functions in the exception monad either do no throw an exception or never return normally.

definition $no\text{-}throw\ P\ A \equiv \{\!|\ P\ |\!\}\ A\ \{\!|\ \lambda\text{-}\ \text{-}.\ True\ |\!\},\{\!|\ \lambda\text{-}\ \text{-}.\ False\ |\!\}$

definition $no\text{-}return\ P\ A \equiv \{\!|\ P\ |\!\}\ A\ \{\!|\lambda\text{-}\ \text{-}.\ False|\!\},\{\!|\lambda\text{-}\ \text{-}.\ True\ |\!\}$

end

theory *NonDetMonadLemmas*
imports *NonDetMonad*
begin

20 General Lemmas Regarding the Nondeterministic State Monad

20.1 Congruence Rules for the Function Package

lemma *bind-cong*[*fundef-cong*]:
$[\![\ f = f';\ \bigwedge v\ s\ s'.\ (v,\ s') \in fst\ (f'\ s) \Longrightarrow g\ v\ s' = g'\ v\ s'\ ]\!] \Longrightarrow f >>= g = f' >>= g'$

apply (*rule ext*)
apply (*auto simp*: *bind-def Let-def split-def intro*: *rev-image-eqI*)
done

lemma *bind-apply-cong* [*fundef-cong*]:
$\llbracket$ $f\ s = f'\ s'$; $\bigwedge rv\ st.\ (rv,\ st) \in fst\ (f'\ s') \Longrightarrow g\ rv\ st = g'\ rv\ st$ $\rrbracket$
$\Longrightarrow (f >>= g)\ s = (f' >>= g')\ s'$
apply (*simp add*: *bind-def*)
apply (*auto simp*: *split-def intro*: *SUP-cong* [*OF refl*] *intro*: *rev-image-eqI*)
done

lemma *bindE-cong*[*fundef-cong*]:
$\llbracket$ $M = M'$; $\bigwedge v\ s\ s'.\ (Inr\ v,\ s') \in fst\ (M'\ s) \Longrightarrow N\ v\ s' = N'\ v\ s'$ $\rrbracket \Longrightarrow bindE\ M\ N = bindE\ M'\ N'$
apply (*simp add*: *bindE-def*)
apply (*rule bind-cong*)
apply (*rule refl*)
apply (*unfold lift-def*)
apply (*case-tac v*, *simp-all*)
done

lemma *bindE-apply-cong*[*fundef-cong*]:
$\llbracket$ $f\ s = f'\ s'$; $\bigwedge rv\ st.\ (Inr\ rv,\ st) \in fst\ (f'\ s') \Longrightarrow g\ rv\ st = g'\ rv\ st$ $\rrbracket$
$\Longrightarrow (f >>=E\ g)\ s = (f' >>=E\ g')\ s'$
apply (*simp add*: *bindE-def*)
apply (*rule bind-apply-cong*)
apply *assumption*
apply (*case-tac rv*, *simp-all add*: *lift-def*)
done

lemma *K-bind-apply-cong*[*fundef-cong*]:
$\llbracket$ $f\ st = f'\ st'$ $\rrbracket \Longrightarrow K\text{-}bind\ f\ arg\ st = K\text{-}bind\ f'\ arg'\ st'$
by *simp*

lemma *when-apply-cong*[*fundef-cong*]:
$\llbracket$ $C = C'$; $s = s'$; $C' \Longrightarrow m\ s' = m'\ s'$ $\rrbracket \Longrightarrow whenE\ C\ m\ s = whenE\ C'\ m'\ s'$
by (*simp add*: *whenE-def*)

lemma *unless-apply-cong*[*fundef-cong*]:
$\llbracket$ $C = C'$; $s = s'$; $\neg\ C' \Longrightarrow m\ s' = m'\ s'$ $\rrbracket \Longrightarrow unlessE\ C\ m\ s = unlessE\ C'\ m'\ s'$
by (*simp add*: *unlessE-def*)

lemma *whenE-apply-cong*[*fundef-cong*]:
$\llbracket$ $C = C'$; $s = s'$; $C' \Longrightarrow m\ s' = m'\ s'$ $\rrbracket \Longrightarrow whenE\ C\ m\ s = whenE\ C'\ m'\ s'$
by (*simp add*: *whenE-def*)

lemma *unlessE-apply-cong*[*fundef-cong*]:
$\llbracket$ $C = C'$; $s = s'$; $\neg\ C' \Longrightarrow m\ s' = m'\ s'$ $\rrbracket \Longrightarrow unlessE\ C\ m\ s = unlessE\ C'\ m'$

s'
by (*simp add*: *unlessE-def*)

20.2 Simplifying Monads

lemma *nested-bind* [*simp*]:
do x <− do y <− f; *return* (*g y*) *od*; *h x od* =
do y <− f; *h* (*g y*) *od*
apply (*clarsimp simp add*: *bind-def*)
apply (*rule ext*)
apply (*clarsimp simp add*: *Let-def split-def return-def*)
done

lemma *fail-bind* [*simp*]:
fail >>= f = *fail*
by (*simp add*: *bind-def fail-def*)

lemma *fail-bindE* [*simp*]:
fail >>=E f = *fail*
by (*simp add*: *bindE-def bind-def fail-def*)

lemma *assert-False* [*simp*]:
assert False >>= f = *fail*
by (*simp add*: *assert-def*)

lemma *assert-True* [*simp*]:
assert True >>= f = *f* ()
by (*simp add*: *assert-def*)

lemma *assertE-False* [*simp*]:
assertE False >>=E f = *fail*
by (*simp add*: *assertE-def*)

lemma *assertE-True* [*simp*]:
assertE True >>=E f = *f* ()
by (*simp add*: *assertE-def*)

lemma *when-False-bind* [*simp*]:
when False g >>= f = *f* ()
by (*rule ext*) (*simp add*: *when-def bind-def return-def*)

lemma *when-True-bind* [*simp*]:
when True g >>= f = *g >>= f*
by (*simp add*: *when-def bind-def return-def*)

lemma *whenE-False-bind* [*simp*]:
whenE False g >>=E f = *f* ()
by (*simp add*: *whenE-def bindE-def returnOk-def lift-def*)

lemma *whenE-True-bind* [*simp*]:
whenE True g >>=E f = *g >>=E f*
by (*simp add*: *whenE-def bindE-def returnOk-def lift-def*)

lemma *when-True* [*simp*]: *when True X* = *X*
by (*clarsimp simp*: *when-def*)

lemma *when-False* [*simp*]: *when False X* = *return* ()
by (*clarsimp simp*: *when-def*)

lemma *unless-False* [*simp*]: *unless False X* = *X*
by (*clarsimp simp*: *unless-def*)

lemma *unlessE-False* [*simp*]: *unlessE False f* = *f*
unfolding *unlessE-def* **by** *fastforce*

lemma *unless-True* [*simp*]: *unless True X* = *return* ()
by (*clarsimp simp*: *unless-def*)

lemma *unlessE-True* [*simp*]: *unlessE True f* = *returnOk* ()
unfolding *unlessE-def* **by** *fastforce*

lemma *unlessE-whenE*:
unlessE P = *whenE* (~*P*)
by (*rule ext*)+ (*simp add*: *unlessE-def whenE-def*)

lemma *unless-when*:
unless P = *when* (~*P*)
by (*rule ext*)+ (*simp add*: *unless-def when-def*)

lemma *gets-to-return* [*simp*]: *gets* (λ*s*. *v*) = *return v*
by (*clarsimp simp*: *gets-def put-def get-def bind-def return-def*)

lemma *assert-opt-Some*:
assert-opt (*Some x*) = *return x*
by (*simp add*: *assert-opt-def*)

lemma *assertE-liftE*:
assertE P = *liftE* (*assert P*)
by (*simp add*: *assertE-def assert-def liftE-def returnOk-def*)

lemma *liftE-handleE′* [*simp*]: ((*liftE a*) <*handle2*> *b*) = *liftE a*
apply (*clarsimp simp*: *liftE-def handleE′-def*)
done

lemma *liftE-handleE* [*simp*]: ((*liftE a*) <*handle*> *b*) = *liftE a*
apply (*unfold handleE-def*)
apply *simp*
done

lemma *condition-split*:
P (*condition* C a b s) = ((((C s) $\longrightarrow$ P (a s)) $\wedge$ ($\neg$ (C s) $\longrightarrow$ P (b s))))
apply (*clarsimp simp*: *condition-def*)
done

lemma *condition-split-asm*:
P (*condition* C a b s) = ($\neg$ (C s $\wedge$ $\neg$ P (a s) $\vee$ $\neg$ C s $\wedge$ $\neg$ P (b s)))
apply (*clarsimp simp*: *condition-def*)
done

lemmas *condition-splits* = *condition-split condition-split-asm*

lemma *condition-true-triv* [*simp*]:
condition (λ-. *True*) A B = A
apply (*rule ext*)
apply (*clarsimp split*: *condition-splits*)
done

lemma *condition-false-triv* [*simp*]:
condition (λ-. *False*) A B = B
apply (*rule ext*)
apply (*clarsimp split*: *condition-splits*)
done

lemma *condition-true*: $\llbracket$ P s $\rrbracket$ $\Longrightarrow$ *condition* P A B s = A s
apply (*clarsimp simp*: *condition-def*)
done

lemma *condition-false*: $\llbracket$ $\neg$ P s $\rrbracket$ $\Longrightarrow$ *condition* P A B s = B s
apply (*clarsimp simp*: *condition-def*)
done

lemmas *arg-cong-bind* = *arg-cong2*[**where** *f*=*bind*]
lemmas *arg-cong-bind1* = *arg-cong-bind*[*OF refl ext*]

21 Low-level monadic reasoning

lemma *monad-eqI* [*intro*]:
$\llbracket$ $\bigwedge r\ t\ s.$ (r, t) $\in$ *fst* (A s) $\Longrightarrow$ (r, t) $\in$ *fst* (B s);
$\bigwedge r\ t\ s.$ (r, t) $\in$ *fst* (B s) $\Longrightarrow$ (r, t) $\in$ *fst* (A s);
$\bigwedge x.$ *snd* (A x) = *snd* (B x) $\rrbracket$
$\Longrightarrow$ (A :: ($'s$, $'a$) *nondet-monad*) = B
apply (*fastforce intro*!: *set-eqI prod-eqI*)
done

lemma *monad-state-eqI* [*intro*]:
$\llbracket$ $\bigwedge r\ t.$ (r, t) $\in$ *fst* (A s) $\Longrightarrow$ (r, t) $\in$ *fst* (B s');
$\bigwedge r\ t.$ (r, t) $\in$ *fst* (B s') $\Longrightarrow$ (r, t) $\in$ *fst* (A s);

$snd\ (A\ s) = snd\ (B\ s')\ \rrbracket$
$\Longrightarrow (A :: ('s, 'a)\ nondet\text{-}monad)\ s = B\ s'$
apply (*fastforce intro*!: *set-eqI prod-eqI*)
done

21.1 General whileLoop reasoning

definition
whileLoop-terminatesE $C\ B \equiv (\lambda r.$
whileLoop-terminates $(\lambda r\ s.\ case\ r\ of\ Inr\ v \Rightarrow C\ v\ s \mid \text{-} \Rightarrow False)\ (lift\ B)\ (Inr\ r))$

lemma *whileLoop-cond-fail*:
$\llbracket \neg\ C\ x\ s\ \rrbracket \Longrightarrow (whileLoop\ C\ B\ x\ s) = (return\ x\ s)$
apply (*auto simp*: *return-def whileLoop-def*
intro: *whileLoop-results.intros*
whileLoop-terminates.intros
elim!: *whileLoop-results.cases*)
done

lemma *whileLoopE-cond-fail*:
$\llbracket \neg\ C\ x\ s\ \rrbracket \Longrightarrow (whileLoopE\ C\ B\ x\ s) = (returnOk\ x\ s)$
apply (*clarsimp simp*: *whileLoopE-def returnOk-def*)
apply (*auto intro*: *whileLoop-cond-fail*)
done

lemma *whileLoop-results-simps-no-move* [*simp*]:
shows $((Some\ x,\ Some\ x) \in whileLoop\text{-}results\ C\ B) = (\neg\ C\ (fst\ x)\ (snd\ x))$
(**is** *?LHS x* = *?RHS x*)
proof (*rule iffI*)
assume *?LHS x*
then have $(\exists\, a.\ Some\ x = Some\ a) \longrightarrow\ ?RHS\ (the\ (Some\ x))$
by (*induct rule*: *whileLoop-results.induct*, *auto*)
thus *?RHS x*
by *clarsimp*
next
assume *?RHS x*
thus *?LHS x*
by (*metis surjective-pairing whileLoop-results.intros*(*1*))
qed

lemma *whileLoop-unroll*:
$(whileLoop\ C\ B\ r) = \ ((condition\ (C\ r)\ (B\ r >>= (whileLoop\ C\ B))\ (return\ r)))$
(**is** *?LHS r* = *?RHS r*)
proof −
have *cond-fail*: $\bigwedge r\ s.\ \neg\ C\ r\ s \Longrightarrow\ ?LHS\ r\ s =\ ?RHS\ r\ s$
apply (*subst whileLoop-cond-fail*, *simp*)
apply (*clarsimp simp*: *condition-def bind-def return-def*)
done

```
  have cond-pass: ⋀r s. C r s ⟹ whileLoop C B r s = (B r >>= (whileLoop C
B)) s
    apply (rule monad-state-eqI)
      apply (clarsimp simp: whileLoop-def bind-def split-def)
      apply (subst (asm) whileLoop-results-simps-valid)
      apply fastforce
     apply (clarsimp simp: whileLoop-def bind-def split-def)
     apply (subst whileLoop-results.simps)
     apply fastforce
    apply (clarsimp simp: whileLoop-def bind-def split-def)
    apply (subst whileLoop-results.simps)
    apply (subst whileLoop-terminates.simps)
    apply fastforce
    done

  show ?thesis
    apply (rule ext)
    apply (metis cond-fail cond-pass condition-def)
    done
qed

lemma whileLoop-unroll':
    (whileLoop C B r) = ((condition (C r) (B r) (return r)) >>= (whileLoop C
B))
  apply (rule ext)
  apply (subst whileLoop-unroll)
  apply (clarsimp simp: condition-def bind-def return-def split-def)
  apply (subst whileLoop-cond-fail, simp)
  apply (clarsimp simp: return-def)
  done

lemma whileLoopE-unroll:
  (whileLoopE C B r) =  ((condition (C r) (B r >>=E (whileLoopE C B))
(returnOk r)))
  apply (rule ext)
  apply (unfold whileLoopE-def)
  apply (subst whileLoop-unroll)
 apply (clarsimp simp: whileLoopE-def bindE-def returnOk-def split: condition-splits)
  apply (clarsimp simp: lift-def)
  apply (rule-tac f=λa. (B r >>= a) x in arg-cong)
  apply (rule ext)+
  apply (clarsimp simp: lift-def split: sum.splits)
  apply (subst whileLoop-unroll)
  apply (subst condition-false)
   apply fastforce
  apply (clarsimp simp: throwError-def)
  done
```

lemma *whileLoopE-unroll′*:
$(whileLoopE\ C\ B\ r) = ((condition\ (C\ r)\ (B\ r)\ (returnOk\ r)) >>=E\ (whileLoopE\ C\ B))$
apply (*rule ext*)
apply (*subst whileLoopE-unroll*)
apply (*clarsimp simp*: *condition-def bindE-def bind-def returnOk-def return-def lift-def split-def*)
apply (*subst whileLoopE-cond-fail*, *simp*)
apply (*clarsimp simp*: *returnOk-def return-def*)
done

lemma *valid-make-schematic-post*:
$(\forall s0.\ \{\!|\ \lambda s.\ P\ s0\ s\ |\!\}\ f\ \{\!|\ \lambda rv\ s.\ Q\ s0\ rv\ s\ |\!\}) \Longrightarrow$
$\{\!|\ \lambda s.\ \exists s0.\ P\ s0\ s \wedge (\forall rv\ s'.\ Q\ s0\ rv\ s' \longrightarrow Q'\ rv\ s')\ |\!\}\ f\ \{\!|\ Q'\ |\!\}$
by (*auto simp add*: *valid-def no-fail-def split*: *prod.splits*)

lemma *validNF-make-schematic-post*:
$(\forall s0.\ \{\!|\ \lambda s.\ P\ s0\ s\ |\!\}\ f\ \{\!|\ \lambda rv\ s.\ Q\ s0\ rv\ s\ |\!\}!) \Longrightarrow$
$\{\!|\ \lambda s.\ \exists s0.\ P\ s0\ s \wedge (\forall rv\ s'.\ Q\ s0\ rv\ s' \longrightarrow Q'\ rv\ s')\ |\!\}\ f\ \{\!|\ Q'\ |\!\}!$
by (*auto simp add*: *valid-def validNF-def no-fail-def split*: *prod.splits*)

lemma *validE-make-schematic-post*:
$(\forall s0.\ \{\!|\ \lambda s.\ P\ s0\ s\ |\!\}\ f\ \{\!|\ \lambda rv\ s.\ Q\ s0\ rv\ s\ |\!\},\ \{\!|\ \lambda rv\ s.\ E\ s0\ rv\ s\ |\!\}) \Longrightarrow$
$\{\!|\ \lambda s.\ \exists s0.\ P\ s0\ s \wedge (\forall rv\ s'.\ Q\ s0\ rv\ s' \longrightarrow Q'\ rv\ s')$
$\wedge (\forall rv\ s'.\ E\ s0\ rv\ s' \longrightarrow E'\ rv\ s')\ |\!\}\ f\ \{\!|\ Q'\ |\!\},\ \{\!|\ E'\ |\!\}$
by (*auto simp add*: *validE-def valid-def no-fail-def split*: *prod.splits sum.splits*)

lemma *validE-NF-make-schematic-post*:
$(\forall s0.\ \{\!|\ \lambda s.\ P\ s0\ s\ |\!\}\ f\ \{\!|\ \lambda rv\ s.\ Q\ s0\ rv\ s\ |\!\},\ \{\!|\ \lambda rv\ s.\ E\ s0\ rv\ s\ |\!\}!) \Longrightarrow$
$\{\!|\ \lambda s.\ \exists s0.\ P\ s0\ s \wedge (\forall rv\ s'.\ Q\ s0\ rv\ s' \longrightarrow Q'\ rv\ s')$
$\wedge (\forall rv\ s'.\ E\ s0\ rv\ s' \longrightarrow E'\ rv\ s')\ |\!\}\ f\ \{\!|\ Q'\ |\!\},\ \{\!|\ E'\ |\!\}!$
by (*auto simp add*: *validE-NF-def validE-def valid-def no-fail-def split*: *prod.splits sum.splits*)

lemma *validNF-conjD1*: $\{\!|\ P\ |\!\}\ f\ \{\!|\ \lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s\ |\!\}! \Longrightarrow \{\!|\ P\ |\!\}\ f\ \{\!|\ Q\ |\!\}!$
by (*fastforce simp*: *validNF-def valid-def no-fail-def*)

lemma *validNF-conjD2*: $\{\!|\ P\ |\!\}\ f\ \{\!|\ \lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s\ |\!\}! \Longrightarrow \{\!|\ P\ |\!\}\ f\ \{\!|\ Q'\ |\!\}!$
by (*fastforce simp*: *validNF-def valid-def no-fail-def*)

end

theory *WP-Pre*
imports
Main
HOL−Eisbach.Eisbach-Tools

begin

named-theorems *wp-pre*

ML ‹
structure WP-Pre = struct

fun append-used-thm thm used-thms = used-thms := !used-thms @ [thm]

fun pre-tac ctxt pre-rules used-ref-option i t = let
fun append-thm used-thm thm =
if Option.isSome used-ref-option
then Seq.map (fn thm => (append-used-thm used-thm (Option.valOf used-ref-option); thm)) thm
else thm;
fun apply-rule t thm = append-thm t (resolve-tac ctxt [t] i thm)
val t2 = FIRST (map apply-rule pre-rules) t |> Seq.hd
val etac = TRY o eresolve-tac ctxt [@{thm FalseE}]
fun dummy-t2 - - = Seq.single t2
val t3 = (dummy-t2 THEN-ALL-NEW etac) i t |> Seq.hd
in if Thm.nprems-of t3 <> Thm.nprems-of t2
then Seq.empty else Seq.single t2 end
handle Option => Seq.empty

fun tac used-ref-option ctxt = let
val pres = Named-Theorems.get ctxt @{named-theorems wp-pre}
in pre-tac ctxt pres used-ref-option end

val method
= Args.context >> (fn - => fn ctxt => Method.SIMPLE-METHOD′ (tac NONE ctxt));
end
›

method-setup *wp-pre0* = ‹*WP-Pre.method*›
method *wp-pre* = *wp-pre0?*

definition
test-wp-pre :: *bool* ⇒ *bool* ⇒ *bool*
where
test-wp-pre P Q = ($P \longrightarrow Q$)

lemma *test-wp-pre-pre*[*wp-pre*]:
test-wp-pre P′ Q ⟹ ($P \Longrightarrow P'$)
⟹ *test-wp-pre P Q*
by (*simp add*: *test-wp-pre-def*)

lemma *demo*:
test-wp-pre P P

apply *wp-pre0+*
 apply (*simp add*: *test-wp-pre-def*, *rule imp-refl*)
apply *simp*
done

end

theory *Datatype-Schematic*

imports
  ../ml−helpers/MLUtils
  ../ml−helpers/TermPatternAntiquote
begin

Introduces a method for improving unification outcomes for schematics with datatype expressions as parameters.

There are two variants: 1. In cases where a schematic is applied to a constant like *True*, we wrap the constant to avoid some undesirable unification candidates.

2. In cases where a schematic is applied to a constructor expression like *Some x* or (x, y), we supply selector expressions like *the* or *fst* to provide more unification candidates. This is only done if parameter that would be selected (e.g. x in *Some x*) contains bound variables which the schematic does not have as parameters.

In the "constructor expression" case, we let users supply additional constructor handlers via the 'datatype$_schematic`attribute.Themethodusesrulesofthefollowingform:$

$\bigwedge x1\ x2\ x3.\ getter\ (constructor\ x1\ x2\ x3) = x2$

These are essentially simp rules for simple "accessor" primrec functions, which are used to turn schematics like

$?P\ (constructor\ x1\ x2\ x3)$

into

$?P'\ x2\ (constructor\ x1\ x2\ x3)$.

ML ‹
  — Anchor used to link error messages back to the documentation above.
  val usage-pos = @{*here*};
›

definition
  ds-id :: $'a \Rightarrow 'a$
where
  ds-id = $(\lambda x.\ x)$

lemma *wrap-ds-id*:
  $x = ds\text{-}id\ x$

by (*simp add*: *ds-id-def*)

ML ‹
structure Datatype-Schematic = struct

fun eq ((idx1, name1, thm1), (idx2, name2, thm2)) =
  idx1 = idx2 andalso
  name1 = name2 andalso
  (Thm.full-prop-of thm1) aconv (Thm.full-prop-of thm2);

structure Datatype-Schematic-Data = Generic-Data
(
 — Keys are names of datatype constructors (like (#)), values are ‘(index, function$_name, thm)$‘.
- ‘function$_name‘isthenameofan”accessor”functionthataccessespartoftheconstructorspecifiedbythekey(soth$
- ‘thm‘ is a theorem showing that the function accesses one of the arguments to the constructor (like *hd* (*?x21.0* # *?x22.0*) = *?x21.0*).
- ‘idx‘ is the index of the constructor argument that the accessor accesses. (eg. since ‘hd‘ accesses the first argument, ‘idx = 0‘; since ‘tl‘ accesses the second argument, ‘idx = 1‘).
  type T = ((int * string * thm) list) Symtab.table;
  val empty = Symtab.empty;
  val extend = I;
  val merge = Symtab.merge-list eq;
);

fun gen-att m =
  Thm.declaration-attribute (fn thm => fn context =>
    Datatype-Schematic-Data.map (m (Context.proof-of context) thm) context);

(* gathers schematic applications from the goal. no effort is made
   to normalise bound variables here, since we'll always be comparing
   elements within a compound application which will be at the same
   level as regards lambdas. *)
fun gather-schem-apps (f $ x) insts = let
    val (f, xs) = strip-comb (f $ x)
    val insts = fold (gather-schem-apps) (f :: xs) insts
  in if is-Var f then (f, xs) :: insts else insts end
  | gather-schem-apps (Abs (-, -, t)) insts
    = gather-schem-apps t insts
  | gather-schem-apps - insts = insts

fun sfirst xs f = get-first f xs

fun get-action ctxt prop = let
    val schem-insts = gather-schem-apps prop [];
    val actions = Datatype-Schematic-Data.get (Context.Proof ctxt);
    fun mk-sel selname T i = let
        val (argTs, resT) = strip-type T
      in Const (selname, resT --> nth argTs i) end

```
  in
    sfirst schem-insts
    (fn (var, xs) => sfirst (Library.tag-list 0 xs)
        (try (fn (idx, x) => let
            val (c, ys) = strip-comb x
            val (fname, T) = dest-Const c
            val acts = Symtab.lookup-list actions fname
            fun interesting arg = not (member Term.aconv-untyped xs arg)
                andalso exists (fn i => not (member (=) xs (Bound i)))
                    (Term.loose-bnos arg)
          in the (sfirst acts (fn (i, selname, thms) => if interesting (nth ys i)
            then SOME (var, idx, mk-sel selname T i, thms) else NONE))
          end)))
  end

fun get-bound-tac ctxt = SUBGOAL (fn (t, i) => case get-action ctxt t of
  SOME (Var ((nm, ix), T), idx, sel, thm) => (fn t => let
    val (argTs, -) = strip-type T
    val ix2 = Thm.maxidx-of t + 1
    val xs = map (fn (i, T) => Free (x ^ string-of-int i, T))
        (Library.tag-list 1 argTs)
    val nx = sel $ nth xs idx
    val v' = Var ((nm, ix2), fastype-of nx --> T)
    val inst-v = fold lambda (rev xs) (betapplys (v' $ nx, xs))
    val t' = Drule.infer-instantiate ctxt
        [((nm, ix), Thm.cterm-of ctxt inst-v)] t
    val t'' = Conv.fconv-rule (Thm.beta-conversion true) t'
  in safe-full-simp-tac (clear-simpset ctxt addsimps [thm]) i t'' end)
  | - => no-tac)

fun id-applicable (f $ x) = let
    val (f, xs) = strip-comb (f $ x)
    val here = is-Var f andalso exists is-Const xs
  in here orelse exists id-applicable (f :: xs) end
  | id-applicable (Abs (-, -, t)) = id-applicable t
  | id-applicable - = false

fun combination-conv cv1 cv2 ct =
  let
    val (ct1, ct2) = Thm.dest-comb ct
    val r1 = SOME (cv1 ct1) handle Option => NONE
    val r2 = SOME (cv2 ct2) handle Option => NONE
    fun mk - (SOME res) = res
      | mk ct NONE = Thm.reflexive ct
  in case (r1, r2) of
      (NONE, NONE) => raise Option
    | - => Thm.combination (mk ct1 r1) (mk ct2 r2)
  end
```

```
val wrap = mk-meta-eq @{thm wrap-ds-id}

fun wrap-const-conv - ct = if is-Const (Thm.term-of ct)
        andalso fastype-of (Thm.term-of ct) <> @{typ unit}
    then Conv.rewr-conv wrap ct
    else raise Option

fun combs-conv conv ctxt ct = case Thm.term-of ct of
    - $ - => combination-conv (combs-conv conv ctxt) (conv ctxt) ct
  | - => conv ctxt ct

fun wrap-conv ctxt ct = case Thm.term-of ct of
    Abs - => Conv.sub-conv wrap-conv ctxt ct
  | f $ x => if is-Var (head-of f) then combs-conv wrap-const-conv ctxt ct
    else if not (id-applicable (f $ x)) then raise Option
    else combs-conv wrap-conv ctxt ct
  | - => raise Option

fun CONVERSION-opt conv i t = CONVERSION conv i t
    handle Option => no-tac t

exception Datatype-Schematic-Error of Pretty.T;

fun apply-pos-markup pos text =
  let
    val props = Position.def-properties-of pos;
    val markup = Markup.properties props (Markup.entity  );
  in Pretty.mark-str (markup, text) end;

fun invalid-accessor ctxt thm : exn =
  Datatype-Schematic-Error ([
    Pretty.str Bad input theorem ',
    Syntax.pretty-term ctxt (Thm.full-prop-of thm),
    Pretty.str '. Click ,
    apply-pos-markup usage-pos *here*,
    Pretty.str  for info on the required rule format. ] |> Pretty.paragraph);

local
  fun dest-accessor' thm =
    case (thm |> Thm.full-prop-of |> HOLogic.dest-Trueprop) of
      @{term-pat ?fun-name ?data-pat = ?rhs} =>
        let
          val fun-name = Term.dest-Const fun-name |> fst;
          val (data-const, data-args) = Term.strip-comb data-pat;
          val data-vars = data-args |> map (Term.dest-Var #> fst);
          val rhs-var = rhs |> Term.dest-Var |> fst;
          val data-name = Term.dest-Const data-const |> fst;
          val rhs-idx = ListExtras.find-index (curry op = rhs-var) data-vars |> the;
        in (fun-name, data-name, rhs-idx) end;
```

```
in
  fun dest-accessor ctxt thm =
    case try dest-accessor' thm of
      SOME x => x
    | NONE => raise invalid-accessor ctxt thm;
end

fun add-rule ctxt thm data =
  let
    val (fun-name, data-name, idx) = dest-accessor ctxt thm;
    val entry = (data-name, (idx, fun-name, thm));
  in Symtab.insert-list eq entry data end;

fun del-rule ctxt thm data =
  let
    val (fun-name, data-name, idx) = dest-accessor ctxt thm;
    val entry = (data-name, (idx, fun-name, thm));
  in Symtab.remove-list eq entry data end;

val add = gen-att add-rule;
val del = gen-att del-rule;

fun wrap-tac ctxt = CONVERSION-opt (wrap-conv ctxt)

fun tac1 ctxt = REPEAT-ALL-NEW (get-bound-tac ctxt) THEN' (TRY o wrap-tac
ctxt)

fun tac ctxt = tac1 ctxt ORELSE' wrap-tac ctxt

val add-section =
  Args.add -- Args.colon >> K (Method.modifier add @{here});

val method =
 Method.sections [add-section] >> (fn - => fn ctxt => Method.SIMPLE-METHOD'
(tac ctxt));

end
›

setup ‹
  Attrib.setup
    @{binding datatype-schematic}
    (Attrib.add-del Datatype-Schematic.add Datatype-Schematic.del)
    Accessor rules to fix datatypes in schematics
›

method-setup datatype-schem = ‹
  Datatype-Schematic.method
›
```

declare *prod.sel*[*datatype-schematic*]
declare *option.sel*[*datatype-schematic*]
declare *list.sel*(*1*,*3*)[*datatype-schematic*]

locale *datatype-schem-demo* **begin**

lemma *handles-nested-constructors*:
  $\exists f. \forall y.$ *f True* (*Some* [*x*, (*y*, *z*)]) = *y*
  apply (*rule exI*, *rule allI*)
  apply *datatype-schem*
  apply (*rule refl*)
  done

datatype *foo* =
    basic nat int
  | *another nat*

primrec *get-basic-0* **where**
  get-basic-0 (*basic x0 x1*) = *x0*

primrec *get-nat* **where**
    get-nat (*basic x -*) = *x*
  | *get-nat* (*another z*) = *z*

lemma *selectively-exposing-datatype-arugments*:
  notes *get-basic-0.simps*[*datatype-schematic*]
  shows $\exists x. \forall a\ b.$ *x* (*basic a b*) = *a*
  apply (*rule exI*, (*rule allI*)+)
  apply *datatype-schem* — Only exposes ‘a‘ to the schematic.
  by (*rule refl*)

lemma *method-handles-primrecs-with-two-constructors*:
  shows $\exists x. \forall a\ b.$ *x* (*basic a b*) = *a*
  apply (*rule exI*, (*rule allI*)+)
  apply (*datatype-schem add*: *get-nat.simps*)
  by (*rule refl*)

end

end

theory *Strengthen*
imports *Main*
begin

Implementation of the *strengthen* tool and the *mk-strg* attribute. See the theory *Strengthen-Demo* for a demonstration.

locale *strengthen-implementation* **begin**

definition *st P rel x y* = (*x* = *y* ∨ (*P* ∧ *rel x y*) ∨ (¬ *P* ∧ *rel y x*))

definition
  st-prop1 :: *prop* ⇒ *prop* ⇒ *prop*
where
  st-prop1 (*PROP P*) (*PROP Q*) ≡ (*PROP Q* ⟹ *PROP P*)

definition
  st-prop2 :: *prop* ⇒ *prop* ⇒ *prop*
where
  st-prop2 (*PROP P*) (*PROP Q*) ≡ (*PROP P* ⟹ *PROP Q*)

definition *failed* == *True*

definition *elim* :: *prop* ⇒ *prop*
where
 elim (*P* :: *prop*) == *P*

definition *oblig* (*P* :: *prop*) == *P*

end

notation *strengthen-implementation.elim* ({*elim*| - |})
notation *strengthen-implementation.oblig* ({*oblig*| - |})
notation *strengthen-implementation.failed* (<*strg−failed*>)

syntax
  -ap-strg-bool :: [′*a*, ′*a*] => ′*a* (- =*strg*<−−|=> -)
  -ap-wkn-bool :: [′*a*, ′*a*] => ′*a* (- =*strg*−−>|=> -)
  -ap-ge-bool :: [′*a*, ′*a*] => ′*a* (- =*strg*<=|=> -)
  -ap-le-bool :: [′*a*, ′*a*] => ′*a* (- =*strg*>=|=> -)

syntax(*xsymbols*)
  -ap-strg-bool :: [′*a*, ′*a*] => ′*a* (- =*strg*⟵|=> -)
  -ap-wkn-bool :: [′*a*, ′*a*] => ′*a* (- =*strg*⟶|=> -)
  -ap-ge-bool :: [′*a*, ′*a*] => ′*a* (- =*strg*≤|=> -)
  -ap-le-bool :: [′*a*, ′*a*] => ′*a* (- =*strg*≥|=> -)

translations
  P =*strg*⟵|=> *Q* == *CONST strengthen-implementation.st* (*CONST False*) (*CONST HOL.implies*) *P Q*
  P =*strg*⟶|=> *Q* == *CONST strengthen-implementation.st* (*CONST True*) (*CONST HOL.implies*) *P Q*
  P =*strg*≤|=> *Q* == *CONST strengthen-implementation.st* (*CONST False*) (*CONST Orderings.less-eq*) *P Q*
 P =*strg*≥|=> *Q* == *CONST strengthen-implementation.st* (*CONST True*) (*CONST Orderings.less-eq*) *P Q*

context *strengthen-implementation* **begin**

lemma *failedI*:
  <*strg−failed*>
  by (*simp add*: *failed-def*)

lemma *strengthen-refl*:
  st P rel x x
  by (*simp add*: *st-def*)

lemma *st-prop-refl*:
  PROP (*st-prop1* (*PROP P*) (*PROP P*))
  PROP (*st-prop2* (*PROP P*) (*PROP P*))
  unfolding *st-prop1-def st-prop2-def*
  by *safe*

lemma *strengthenI*:
  rel x y $\Longrightarrow$ *st True rel x y*
  rel y x $\Longrightarrow$ *st False rel x y*
  by (*simp-all add*: *st-def*)

lemmas *imp-to-strengthen* = *strengthenI*(*2*)[**where** *rel*=($\longrightarrow$)]
lemmas *rev-imp-to-strengthen* = *strengthenI*(*1*)[**where** *rel*=($\longrightarrow$)]
lemmas *ord-to-strengthen* = *strengthenI*[**where** *rel*=($\le$)]

lemma *use-strengthen-imp*:
  st False ($\longrightarrow$) *Q P* $\Longrightarrow$ *P* $\Longrightarrow$ *Q*
  by (*simp add*: *st-def*)

lemma *use-strengthen-prop-elim*:
  PROP P $\Longrightarrow$ *PROP* (*st-prop2* (*PROP P*) (*PROP Q*))
    $\Longrightarrow$ (*PROP Q* $\Longrightarrow$ *PROP R*) $\Longrightarrow$ *PROP R*
  unfolding *st-prop2-def*
  apply (*drule*(*1*) *meta-mp*)+
  apply *assumption*
  done

lemma *strengthen-Not*:
  st False rel x y $\Longrightarrow$ *st* ($\neg$ *True*) *rel x y*
  st True rel x y $\Longrightarrow$ *st* ($\neg$ *False*) *rel x y*
  by *auto*

lemmas *gather* =
    swap-prems-eq[**where** *A*=*PROP* (*Trueprop P*) **and** *B*=*PROP* (*elim Q*) **for** *P Q*]
    swap-prems-eq[**where** *A*=*PROP* (*Trueprop P*) **and** *B*=*PROP* (*oblig Q*) **for** *P Q*]

lemma *mk-True-imp*:
  $P \equiv True \longrightarrow P$
  by *simp*

lemma *narrow-quant*:
  $(\bigwedge x.\ PROP\ P \Longrightarrow PROP\ (Q\ x)) \equiv (PROP\ P \Longrightarrow (\bigwedge x.\ PROP\ (Q\ x)))$
  $(\bigwedge x.\ (R \longrightarrow S\ x)) \equiv PROP\ (Trueprop\ (R \longrightarrow (\forall x.\ S\ x)))$
  $(\bigwedge x.\ (S\ x \longrightarrow R)) \equiv PROP\ (Trueprop\ ((\exists x.\ S\ x) \longrightarrow R))$
  apply (*simp-all add*: *atomize-all*)
  apply *rule*
   apply *assumption*
  apply *assumption*
  done

ML ‹
```
structure Make-Strengthen-Rule = struct

fun binop-conv' cv1 cv2 = Conv.combination-conv (Conv.arg-conv cv1) cv2;

val mk-elim = Conv.rewr-conv @{thm elim-def[symmetric]}
val mk-oblig = Conv.rewr-conv @{thm oblig-def[symmetric]}

fun count-vars t = Term.fold-aterms
    (fn (Var v) => Termtab.map-default (Var v, 0) (fn x => x + 1)
        | - => I) t Termtab.empty

fun gather-to-imp ctxt drule pattern = let
    val pattern = (if drule then D :: pattern else pattern)
    fun inner pat ct = case (head-of (Thm.term-of ct), pat) of
        (@{term Pure.imp}, (E :: pat)) => binop-conv' mk-elim (inner pat) ct
      | (@{term Pure.imp}, (A :: pat)) => binop-conv' mk-elim (inner pat) ct
      | (@{term Pure.imp}, (O :: pat)) => binop-conv' mk-oblig (inner pat) ct
       | (@{term Pure.imp}, -) => binop-conv' (Object-Logic.atomize ctxt) (inner
(drop 1 pat)) ct
      | (-, []) => Object-Logic.atomize ctxt ct
      | (-, pat) => raise THM (gather-to-imp: leftover pattern: ^ commas pat, 1,
[])
    fun simp thms = Raw-Simplifier.rewrite ctxt false thms
    fun ensure-imp ct = case strip-comb (Thm.term-of ct) |> apsnd (map head-of)
     of
        (@{term Pure.imp}, -) => Conv.arg-conv ensure-imp ct
      | (@{term HOL.Trueprop}, [@{term HOL.implies}]) => Conv.all-conv ct
       | (@{term HOL.Trueprop}, -) => Conv.arg-conv (Conv.rewr-conv @{thm
mk-True-imp}) ct
      | - => raise CTERM (gather-to-imp, [ct])
    val gather = simp @{thms gather}
        then-conv (if drule then Conv.all-conv else simp @{thms atomize-conjL})
        then-conv simp @{thms atomize-imp}
        then-conv ensure-imp
```

```
  in Conv.fconv-rule (inner pattern then-conv gather) end

fun imp-list t = let
    val (x, y) = Logic.dest-implies t
  in x :: imp-list y end handle TERM - => [t]

fun mk-ex (xnm, T) t = HOLogic.exists-const T $ Term.lambda (Var (xnm, T))
t
fun mk-all (xnm, T) t = HOLogic.all-const T $ Term.lambda (Var (xnm, T)) t

fun quantify-vars ctxt drule thm = let
    val (lhs, rhs) = Thm.concl-of thm |> HOLogic.dest-Trueprop
      |> HOLogic.dest-imp
    val all-vars = count-vars (Thm.prop-of thm)
    val new-vars = count-vars (if drule then rhs else lhs)
     val quant = filter (fn v => Termtab.lookup new-vars v = Termtab.lookup
all-vars v)
            (Termtab.keys new-vars)
        |> map (Thm.cterm-of ctxt)
  in fold Thm.forall-intr quant thm
    |> Conv.fconv-rule (Raw-Simplifier.rewrite ctxt false @{thms narrow-quant})
  end

fun mk-strg (typ, pat) ctxt thm = let
    val drule = typ = D orelse typ = D'
    val imp = gather-to-imp ctxt drule pat thm
      |> (if typ = I' orelse typ = D'
          then quantify-vars ctxt drule else I)
  in if typ = I orelse typ = I'
    then imp RS @{thm imp-to-strengthen}
    else if drule then imp RS @{thm rev-imp-to-strengthen}
    else if typ = lhs then imp RS @{thm ord-to-strengthen(1)}
    else if typ = rhs then imp RS @{thm ord-to-strengthen(2)}
    else raise THM (mk-strg: unknown type:  ^ typ, 1, [thm])
 end

fun auto-mk ctxt thm = let
    val concl-C = try (fst o dest-Const o head-of
        o HOLogic.dest-Trueprop) (Thm.concl-of thm)
  in case (Thm.nprems-of thm, concl-C) of
    (-, SOME @{const-name failed}) => thm
  | (-, SOME @{const-name st}) => thm
  | (0, SOME @{const-name HOL.implies}) => (thm RS @{thm imp-to-strengthen}
      handle THM - => @{thm failedI})
  | - => mk-strg (I', []) ctxt thm
  end

fun mk-strg-args (SOME (typ, pat)) ctxt thm = mk-strg (typ, pat) ctxt thm
  | mk-strg-args NONE ctxt thm = auto-mk ctxt thm
```

val arg-pars = *Scan.option* (*Scan.first* (*map Args*.$$$ [*I*, *I′*, *D*, *D′*, *lhs*, *rhs*])
  −− *Scan.repeat* (*Args*.$$$ *A* || *Args*.$$$ *E* || *Args*.$$$ *O* || *Args*.$$$ -))

val attr-pars : *attribute context-parser*
    = (*Scan.lift arg-pars* −− *Args.context*)
        >> (*fn* (*args*, *ctxt*) => *Thm.rule-attribute* [] (*K* (*mk-strg-args args ctxt*)))

end
›

end

attribute-setup *mk-strg* = ‹*Make-Strengthen-Rule.attr-pars*›
          put rule in 'strengthen' form (*see theory Strengthen-Demo*)

Quick test.

lemmas *foo* = *nat.induct*[*mk-strg I O O*]
    nat.induct[*mk-strg D O*]
    nat.induct[*mk-strg I′ E*]
    exI[*mk-strg I′*] *exI*[*mk-strg I*]

context *strengthen-implementation* **begin**

lemma *do-elim*:
  PROP P ⟹ *PROP elim* (*PROP P*)
  by (*simp add*: *elim-def*)

lemma *intro-oblig*:
  PROP P ⟹ *PROP oblig* (*PROP P*)
  by (*simp add*: *oblig-def*)

ML ‹

structure Strengthen = *struct*

structure Congs = *Theory-Data*
(*struct*
    type T = *thm list*
    val empty = []
    val extend = *I*
    val merge = *Thm.merge-thms*;
end);

val tracing = *Attrib.config-bool* @{*binding strengthen-trace*} (*K false*)

fun map-context-total f (*Context.Theory t*) = (*Context.Theory* (*f t*))
  | *map-context-total f* (*Context.Proof p*)

```
  = (Context.Proof (Context.raw-transfer (f (Proof-Context.theory-of p)) p))

val strg-add = Thm.declaration-attribute
      (fn thm => map-context-total (Congs.map (Thm.add-thm thm)));

val strg-del = Thm.declaration-attribute
      (fn thm => map-context-total (Congs.map (Thm.del-thm thm)));

val setup =
  Attrib.setup @{binding strg} (Attrib.add-del strg-add strg-del)
    strengthening congruence rules
    #> snd tracing;

fun goal-predicate t = let
    val gl = Logic.strip-assums-concl t
    val cn = head-of #> dest-Const #> fst
  in if cn gl = @{const-name oblig} then oblig
    else if cn gl = @{const-name elim} then elim
    else if cn gl = @{const-name st-prop1} then st-prop1
    else if cn gl = @{const-name st-prop2} then st-prop2
    else if cn (HOLogic.dest-Trueprop gl) = @{const-name st} then st
    else
  end handle TERM - =>

fun do-elim ctxt = SUBGOAL (fn (t, i) => if goal-predicate t = elim
    then eresolve-tac ctxt @{thms do-elim} i else all-tac)

fun final-oblig-strengthen ctxt = SUBGOAL (fn (t, i) => case goal-predicate t of
    oblig => resolve-tac ctxt @{thms intro-oblig} i
  | st => resolve-tac ctxt @{thms strengthen-refl} i
  | st-prop1 => resolve-tac ctxt @{thms st-prop-refl} i
  | st-prop2 => resolve-tac ctxt @{thms st-prop-refl} i
  | - => all-tac)

infix 1 THEN-TRY-ALL-NEW;

(* Like THEN-ALL-NEW but allows failure, although at least one subsequent
   method must succeed. *)
fun (tac1 THEN-TRY-ALL-NEW tac2) i st = let
    fun inner b j st = if i > j then (if b then all-tac else no-tac) st
      else ((tac2 j THEN inner true (j − 1)) ORELSE inner b (j − 1)) st
  in st |> (tac1 i THEN (fn st' =>
    inner false (i + Thm.nprems-of st' − Thm.nprems-of st) st')) end

fun maybe-trace-tac false - - = K all-tac
  | maybe-trace-tac true ctxt msg = SUBGOAL (fn (t, -) => let
    val tr = Pretty.big-list msg [Syntax.pretty-term ctxt t]
  in
    Pretty.writeln tr;
```

```
    all-tac
  end)

fun maybe-trace-rule false - - rl = rl
  | maybe-trace-rule true ctxt msg rl = let
    val tr = Pretty.big-list msg [Syntax.pretty-term ctxt (Thm.prop-of rl)]
  in
    Pretty.writeln tr;
    rl
  end

type params = {trace : bool, once : bool}

fun params once ctxt = {trace = Config.get ctxt (fst tracing), once = once}

fun apply-tac-as-strg ctxt (params : params) (tac : tactic)
  = SUBGOAL (fn (t, i) => case Logic.strip-assums-concl t of
      @{term Trueprop} $ (@{term st False (⟶)} $ x $ -)
      => let
    val triv = Thm.trivial (Thm.cterm-of ctxt (HOLogic.mk-Trueprop x))
    val trace = #trace params
  in
    fn thm => tac triv
        |> Seq.map (maybe-trace-rule trace ctxt apply-tac-as-strg: making strg)
        |> Seq.maps (Seq.try (Make-Strengthen-Rule.auto-mk ctxt))
        |> Seq.maps (fn str-rl => resolve-tac ctxt [str-rl] i thm)
  end | - => no-tac)

fun opt-tac f (SOME v) = f v
  | opt-tac - NONE = K no-tac

fun apply-strg ctxt (params : params) congs rules tac = EVERY′ [
    maybe-trace-tac (#trace params) ctxt apply-strg,
    DETERM o TRY o resolve-tac ctxt @{thms strengthen-Not},
    DETERM o ((resolve-tac ctxt rules THEN-ALL-NEW do-elim ctxt)
        ORELSE′ (opt-tac (apply-tac-as-strg ctxt params) tac)
        ORELSE′ (resolve-tac ctxt congs THEN-TRY-ALL-NEW
            (fn i => apply-strg ctxt params congs rules tac i)))
]

fun setup-strg ctxt params thms meths = let
    val congs = Congs.get (Proof-Context.theory-of ctxt)
    val rules = map (Make-Strengthen-Rule.auto-mk ctxt) thms
    val tac = case meths of [] => NONE
      | - => SOME (FIRST (map (fn meth => Method.NO-CONTEXT-TACTIC
ctxt
        (Method.evaluate meth ctxt [])) meths))
  in apply-strg ctxt params congs rules tac
        THEN-ALL-NEW final-oblig-strengthen ctxt end
```

```
fun strengthen ctxt asm concl thms meths = let
    val strg = setup-strg ctxt (params false ctxt) thms meths
  in
    (if not concl then K no-tac
        else resolve-tac ctxt @{thms use-strengthen-imp} THEN' strg)
    ORELSE' (if not asm then K no-tac
        else eresolve-tac ctxt @{thms use-strengthen-prop-elim} THEN' strg)
  end

fun default-strengthen ctxt thms = strengthen ctxt false true thms []

val strengthen-args =
  Attrib.thms >> curry (fn (rules, ctxt) =>
    Method.CONTEXT-METHOD (fn - =>
      Method.RUNTIME (Method.CONTEXT-TACTIC
        (strengthen ctxt false true rules [] 1))
    )
  );

val strengthen-asm-args =
  Attrib.thms >> curry (fn (rules, ctxt) =>
    Method.CONTEXT-METHOD (fn - =>
      Method.RUNTIME (Method.CONTEXT-TACTIC
        (strengthen ctxt true false rules [] 1))
    )
  );

val strengthen-method-args =
  Method.text-closure >> curry (fn (meth, ctxt) =>
    Method.CONTEXT-METHOD (fn - =>
      Method.RUNTIME (Method.CONTEXT-TACTIC
        (strengthen ctxt true true [] [meth] 1))
    )
  );

end

›
```

end

setup *Strengthen.setup*

method-setup *strengthen* = ‹*Strengthen.strengthen-args*›
strengthen the goal (*see theory Strengthen-Demo*)

method-setup *strengthen-asm* = ‹*Strengthen.strengthen-asm-args*›
apply ''strengthen'' to weaken an assumption

method-setup *strengthen-method* = ⟨*Strengthen.strengthen-method-args*⟩
use an argument method in ''strengthen'' sites

Important strengthen congruence rules.

context *strengthen-implementation* **begin**

lemma *strengthen-imp-imp*[*simp*]:
$st\ True\ (\longrightarrow)\ A\ B = (A \longrightarrow B)$
$st\ False\ (\longrightarrow)\ A\ B = (B \longrightarrow A)$
by (*simp-all add*: *st-def*)

abbreviation(*input*)
$st\text{-}ord\ t \equiv st\ t\ ((\leq) :: ('a :: preorder) \Rightarrow \text{-})$

lemma *strengthen-imp-ord*[*simp*]:
$st\text{-}ord\ True\ A\ B = (A \leq B)$
$st\text{-}ord\ False\ A\ B = (B \leq A)$
by (*auto simp add*: *st-def*)

lemma *strengthen-imp-conj* [*strg*]:
$[\![\ A' \Longrightarrow st\ F\ (\longrightarrow)\ B\ B';\ B \Longrightarrow st\ F\ (\longrightarrow)\ A\ A'\ ]\!]$
$\Longrightarrow st\ F\ (\longrightarrow)\ (A \wedge B)\ (A' \wedge B')$
by (*cases F*, *auto*)

lemma *strengthen-imp-disj* [*strg*]:
$[\![\ \neg\ A' \Longrightarrow st\ F\ (\longrightarrow)\ B\ B';\ \neg\ B \Longrightarrow st\ F\ (\longrightarrow)\ A\ A'\ ]\!]$
$\Longrightarrow st\ F\ (\longrightarrow)\ (A \vee B)\ (A' \vee B')$
by (*cases F*, *auto*)

lemma *strengthen-imp-implies* [*strg*]:
$[\![\ st\ (\neg\ F)\ (\longrightarrow)\ X\ X';\ X \Longrightarrow st\ F\ (\longrightarrow)\ Y\ Y'\ ]\!]$
$\Longrightarrow st\ F\ (\longrightarrow)\ (X \longrightarrow Y)\ (X' \longrightarrow Y')$
by (*cases F*, *auto*)

lemma *strengthen-all*[*strg*]:
$[\![\ \bigwedge x.\ st\ F\ (\longrightarrow)\ (P\ x)\ (Q\ x)\ ]\!]$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\forall x.\ P\ x)\ (\forall x.\ Q\ x)$
by (*cases F*, *auto*)

lemma *strengthen-ex*[*strg*]:
$[\![\ \bigwedge x.\ st\ F\ (\longrightarrow)\ (P\ x)\ (Q\ x)\ ]\!]$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\exists x.\ P\ x)\ (\exists x.\ Q\ x)$
by (*cases F*, *auto*)

lemma *strengthen-Ball*[*strg*]:
$[\![\ st\text{-}ord\ (Not\ F)\ S\ S';$
$\bigwedge x.\ x \in S \Longrightarrow st\ F\ (\longrightarrow)\ (P\ x)\ (Q\ x)\ ]\!]$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\forall x \in S.\ P\ x)\ (\forall x \in S'.\ Q\ x)$

by (*cases* F, *auto*)

lemma *strengthen-Bex*[*strg*]:
$[\![$ *st-ord* F S S';
$\bigwedge x.\ x \in S \Longrightarrow$ *st* F $(\longrightarrow)$ $(P\ x)$ $(Q\ x)$ $]\!]$
$\Longrightarrow$ *st* F $(\longrightarrow)$ $(\exists x \in S.\ P\ x)$ $(\exists x \in S'.\ Q\ x)$
by (*cases* F, *auto*)

lemma *strengthen-Collect*[*strg*]:
$[\![\bigwedge x.$ *st* F $(\longrightarrow)$ $(P\ x)$ $(P'\ x)$ $]\!]$
$\Longrightarrow$ *st-ord* F $\{x.\ P\ x\}$ $\{x.\ P'\ x\}$
by (*cases* F, *auto*)

lemma *strengthen-mem*[*strg*]:
$[\![$ *st-ord* F S S' $]\!]$
$\Longrightarrow$ *st* F $(\longrightarrow)$ $(x \in S)$ $(x \in S')$
by (*cases* F, *auto*)

lemma *strengthen-ord*[*strg*]:
st-ord $(\neg\ F)$ x x' $\Longrightarrow$ *st-ord* F y y'
$\Longrightarrow$ *st* F $(\longrightarrow)$ $(x \leq y)$ $(x' \leq y')$
by (*cases* F, *simp-all*, (*metis order-trans*)+)

lemma *strengthen-strict-ord*[*strg*]:
st-ord $(\neg\ F)$ x x' $\Longrightarrow$ *st-ord* F y y'
$\Longrightarrow$ *st* F $(\longrightarrow)$ $(x < y)$ $(x' < y')$
by (*cases* F, *simp-all*, (*metis order-le-less-trans order-less-le-trans*)+)

lemma *strengthen-image*[*strg*]:
st-ord F S S' $\Longrightarrow$ *st-ord* F $(f$ ‘ $S)$ $(f$ ‘ $S')$
by (*cases* F, *auto*)

lemma *strengthen-vimage*[*strg*]:
st-ord F S S' $\Longrightarrow$ *st-ord* F $(f$ −‘ $S)$ $(f$ −‘ $S')$
by (*cases* F, *auto*)

lemma *strengthen-Int*[*strg*]:
st-ord F A A' $\Longrightarrow$ *st-ord* F B B' $\Longrightarrow$ *st-ord* F $(A \cap B)$ $(A' \cap B')$
by (*cases* F, *auto*)

lemma *strengthen-Un*[*strg*]:
st-ord F A A' $\Longrightarrow$ *st-ord* F B B' $\Longrightarrow$ *st-ord* F $(A \cup B)$ $(A' \cup B')$
by (*cases* F, *auto*)

lemma *strengthen-UN*[*strg*]:
st-ord F A A' $\Longrightarrow$ $(\bigwedge x.\ x \in A \Longrightarrow$ *st-ord* F $(B\ x)$ $(B'\ x))$
$\Longrightarrow$ *st-ord* F $(\bigcup x \in A.\ B\ x)$ $(\bigcup x \in A'.\ B'\ x)$
by (*cases* F, *auto*)

lemma *strengthen-INT*[*strg*]:
  st-ord $(\neg\ F)\ A\ A' \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow$ *st-ord* $F\ (B\ x)\ (B'\ x))$
    $\Longrightarrow$ *st-ord* $F\ (\bigcap x \in A.\ B\ x)\ (\bigcap x \in A'.\ B'\ x)$
  by (*cases* F, *auto*)

lemma *strengthen-imp-strengthen-prop*[*strg*]:
  st False $(\longrightarrow)\ P\ Q \Longrightarrow$ *PROP* (*st-prop1* (*Trueprop* P) (*Trueprop* Q))
  st True $(\longrightarrow)\ P\ Q \Longrightarrow$ *PROP* (*st-prop2* (*Trueprop* P) (*Trueprop* Q))
  unfolding *st-prop1-def st-prop2-def*
  by *auto*

lemma *st-prop-meta-imp*[*strg*]:
  PROP (*st-prop2* (*PROP* X) (*PROP* X'))
    $\Longrightarrow$ *PROP* (*st-prop1* (*PROP* Y) (*PROP* Y'))
    $\Longrightarrow$ *PROP* (*st-prop1* (*PROP* $X \Longrightarrow$ *PROP* Y) (*PROP* $X' \Longrightarrow$ *PROP* Y'))
  PROP (*st-prop1* (*PROP* X) (*PROP* X'))
    $\Longrightarrow$ *PROP* (*st-prop2* (*PROP* Y) (*PROP* Y'))
    $\Longrightarrow$ *PROP* (*st-prop2* (*PROP* $X \Longrightarrow$ *PROP* Y) (*PROP* $X' \Longrightarrow$ *PROP* Y'))
  unfolding *st-prop1-def st-prop2-def*
  by (*erule meta-mp* | *assumption*)+

lemma *st-prop-meta-all*[*strg*]:
  $(\bigwedge x.$ *PROP* (*st-prop1* (*PROP* $(X\ x)$) (*PROP* $(X'\ x)$)))
    $\Longrightarrow$ *PROP* (*st-prop1* $(\bigwedge x.$ *PROP* $(X\ x))$ $(\bigwedge x.$ *PROP* $(X'\ x)))$
  $(\bigwedge x.$ *PROP* (*st-prop2* (*PROP* $(X\ x)$) (*PROP* $(X'\ x)$)))
    $\Longrightarrow$ *PROP* (*st-prop2* $(\bigwedge x.$ *PROP* $(X\ x))$ $(\bigwedge x.$ *PROP* $(X'\ x)))$
  unfolding *st-prop1-def st-prop2-def*
   apply (*rule Pure.asm-rl*)
   apply (*erule meta-allE*, *erule meta-mp*)
   apply *assumption*
  apply (*rule Pure.asm-rl*)
  apply (*erule meta-allE*, *erule meta-mp*)
  apply *assumption*
  done

end

lemma *imp-consequent*:
  $P \longrightarrow Q \longrightarrow P$ **by** *simp*

Test cases.

lemma
  assumes x: $\bigwedge x.\ P\ x \longrightarrow Q\ x$
  shows $\{x.\ x \neq None \wedge P\ (the\ x)\} \subseteq \{y.\ \forall x.\ y = Some\ x \longrightarrow Q\ x\}$
  apply (*strengthen* x)
  apply *clarsimp*
  done

locale *strengthen-silly-test* **begin**

definition
  silly :: *nat* ⇒ *nat* ⇒ *bool*
where
  silly x y = (*x* ≤ *y*)

lemma *silly-trans*:
  silly x y ⟹ *silly y z* ⟹ *silly x z*
  by (*simp add*: *silly-def*)

lemma *silly-refl*:
  silly x x
  by (*simp add*: *silly-def*)

lemma *foo*:
  silly x y ⟹ *silly a b* ⟹ *silly b c*
    ⟹ *silly x y* ∧ (∀ *x* :: *nat*. *silly a c*)
  using [[*strengthen-trace* = *true*]]
  apply (*strengthen silly-trans*[*mk-strg I E*])+
  apply (*strengthen silly-refl*)
  apply *simp*
  done

lemma *foo-asm*:
  silly x y ⟹ *silly y z*
    ⟹ (*silly x z* ⟹ *silly a b*) ⟹ *silly z z* ⟹ *silly a b*
  apply (*strengthen-asm silly-trans*[*mk-strg I A*])
  apply (*strengthen-asm silly-trans*[*mk-strg I A*])
  apply *simp*
  done

lemma *foo-method*:
  silly x y ⟹ *silly a b* ⟹ *silly b c*
    ⟹ *silly x y* ∧ (∀ *x* :: *nat*. *z* ⟶ *silly a c*)
  using [[*strengthen-trace* = *true*]]
  apply *simp*
  apply (*strengthen-method* ‹*rule silly-trans*›)
  apply (*strengthen-method* ‹*rule exI*[*where x*=*b*]›)
  apply *simp*
  done

end
end

theory *WPFix*

imports

../Datatype-Schematic
../Strengthen

begin

WPFix handles four issues which are annoying with precondition schematics: 1. Schematics in obligation (postcondition) positions which remain unset after goals are solved. They should be instantiated to True. 2. Schematics which appear in multiple precondition positions. They should be instantiated to a conjunction and then separated. 3/4. Schematics applied to datatype expressions such as *True* or *Some x*. for details.

lemma *use-strengthen-prop-intro*:
PROP P ⟹ *PROP* (*strengthen-implementation.st-prop1* (*PROP Q*) (*PROP P*))
⟹ *PROP Q*
unfolding *strengthen-implementation.st-prop1-def*
apply (*drule*(*1*) *meta-mp*)+
apply *assumption*
done

definition
target-var :: *int* ⇒ *'a* ⇒ *'a*
where
target-var n x = *x*

lemma *strengthen-to-conjunct1-target*:
strengthen-implementation.st True (⟶)
(*target-var n* (*P* ∧ *Q*)) (*target-var n P*)
by (*simp add*: *strengthen-implementation.st-def target-var-def*)

lemma *strengthen-to-conjunct2-target-trans*:
strengthen-implementation.st True (⟶)
(*target-var n Q*) *R*
⟹ *strengthen-implementation.st True* (⟶)
(*target-var n* (*P* ∧ *Q*)) *R*
by (*simp add*: *strengthen-implementation.st-def target-var-def*)

lemma *target-var-drop-func*:
target-var n f = (λ*x*. *target-var n* (*f x*))
by (*simp add*: *target-var-def*)

named-theorems *wp-fix-strgs*

lemma *strg-target-to-true*:
strengthen-implementation.st F (⟶) (*target-var n True*) *True*
by (*simp add*: *target-var-def strengthen-implementation.strengthen-refl*)

ML ‹
structure WPFix = *struct*

```
val st-refl = @{thm strengthen-implementation.strengthen-refl}
val st-refl-True = @{thm strengthen-implementation.strengthen-refl[where x=True]}
val st-refl-target-True = @{thm strg-target-to-true}
val st-refl-non-target
   = @{thm strengthen-implementation.strengthen-refl[where x=target-var (−1) v
for v]}

val conv-to-target = mk-meta-eq @{thm target-var-def[symmetric]}

val tord = Term-Ord.fast-term-ord
fun has-var vars t = not (null (Ord-List.inter tord vars
       (Ord-List.make tord (map Var (Term.add-vars t [])))))

fun get-vars prop = map Var (Term.add-vars prop [])
   |> Ord-List.make tord
   |> filter (fn v => snd (strip-type (fastype-of v)) = HOLogic.boolT)

val st-intro = @{thm use-strengthen-prop-intro}
val st-not = @{thms strengthen-implementation.strengthen-Not}
val st-conj2-trans = @{thm strengthen-to-conjunct2-target-trans}
val st-conj1 = @{thm strengthen-to-conjunct1-target}

(* assumes Strengthen.goal-predicate g is st *)
fun dest-strg g = case Strengthen.goal-predicate g of
   st => (case HOLogic.dest-Trueprop (Logic.strip-assums-concl g) of
       (Const - $ mode $ rel $ lhs $ rhs) => (st, SOME (mode, rel, lhs, rhs))
     | - => error (dest-strg  ^ @{make-string} g)
   )
 | nm => (nm, NONE)

fun get-target (Const (@{const-name target-var}, -) $ n $ -)
 = (try (HOLogic.dest-number #> snd) n)
 | get-target - = NONE

fun is-target P t = case get-target t of NONE => false
 | SOME v => P v

fun is-target-head P (f $ v) = is-target P (f $ v) orelse is-target-head P f
 | is-target-head - - = false

fun has-target P (f $ v) = is-target P (f $ v)
   orelse has-target P f orelse has-target P v
 | has-target P (Abs (-, -, t)) = has-target P t
 | has-target - - = false

fun apply-strgs congs ctxt = SUBGOAL (fn (t, i) => case
       dest-strg t of
   (st-prop1, -) => resolve-tac ctxt congs i
```

```
  | (st-prop2, -) => resolve-tac ctxt congs i
  | (st, SOME (-, -, lhs, -)) => resolve-tac ctxt st-not i
    ORELSE eresolve-tac ctxt [thin-rl] i
    ORELSE resolve-tac ctxt [st-refl-non-target] i
    ORELSE (if is-target-head (fn v => v >= 0) lhs
        then no-tac
        else if not (has-target (fn v => v >= 0) lhs)
        then resolve-tac ctxt [st-refl] i
        else if is-Const (head-of lhs)
        then (resolve-tac ctxt congs i ORELSE resolve-tac ctxt [st-refl] i)
        else resolve-tac ctxt [st-refl] i
    )
  | - => no-tac
  )

fun strg-proc ctxt = let
    val congs1 = Named-Theorems.get ctxt @{named-theorems wp-fix-strgs}
    val thy = Proof-Context.theory-of ctxt
    val congs2 = Strengthen.Congs.get thy
    val strg = apply-strgs (congs1 @ congs2) ctxt
  in REPEAT-ALL-NEW strg end

fun target-var-conv vars ctxt ct = case Thm.term-of ct of
    Abs - => Conv.sub-conv (target-var-conv vars) ctxt ct
  | Var v => Conv.rewr-conv (Drule.infer-instantiate ctxt
        [((n, 1), Thm.cterm-of ctxt (HOLogic.mk-number @{typ int}
            (find-index (fn v2 => v2 = Var v) vars)))] conv-to-target) ct
  | - $ - => Datatype-Schematic.combs-conv (target-var-conv vars) ctxt ct
  | - => raise Option

fun st-intro-tac ctxt = CSUBGOAL (fn (ct, i) => fn thm => let
        val intro = Drule.infer-instantiate ctxt [((Q, 0), ct)]
          (Thm.incr-indexes (Thm.maxidx-of thm + 1) st-intro)
      in compose-tac ctxt (false, intro, 2) i
      end thm)

fun intro-tac ctxt vs = SUBGOAL (fn (t, i) => if has-var vs t
    then CONVERSION (target-var-conv vs ctxt) i
        THEN CONVERSION (Simplifier.full-rewrite (clear-simpset ctxt
            addsimps @{thms target-var-drop-func}
        )) i
        THEN st-intro-tac ctxt i
    else all-tac)

fun classify v thm = let
    val has-t = has-target (fn v' => v' = v)
    val relevant = filter (has-t o fst)
        (Thm.prems-of thm ~~ (1 upto Thm.nprems-of thm))
        |> map (apfst (Logic.strip-assums-concl #> Envir.beta-eta-contract))
```

```
    fun class t = case dest-strg t of
        (st, SOME (@{term True}, @{term (-->)}, lhs, -))
            => if has-t lhs then SOME true else NONE
      | (st, SOME (@{term False}, @{term (-->)}, lhs, -))
            => if has-t lhs then SOME false else NONE
      | - => NONE
    val classn = map (apfst class) relevant
    fun get k = map snd (filter (fn (k', -) => k' = k) classn)
  in if (null relevant) then NONE
    else if not (null (get NONE))
    then NONE
    else if null (get (SOME true))
    then SOME (to-true, map snd relevant)
    else if length (get (SOME true)) > 1
    then SOME (to-conj, get (SOME true))
    else NONE
  end

fun ONGOALS tac is = let
    val is = rev (sort int-ord is)
  in EVERY (map tac is) end

fun act-on ctxt (to-true, is)
    = ONGOALS (resolve-tac ctxt [st-refl-target-True]) is
  | act-on ctxt (to-conj, is)
    = ONGOALS (resolve-tac ctxt [st-conj2-trans]) (drop 1 is)
      THEN (if length is > 2 then act-on ctxt (to-conj, drop 1 is)
        else ONGOALS (resolve-tac ctxt [st-refl]) (drop 1 is))
      THEN ONGOALS (resolve-tac ctxt [st-conj1]) (take 1 is)
  | act-on - (s, -) = error (act-on: ^ s)

fun act ctxt check vs thm = let
    val acts = map-filter (fn v => classify v thm) vs
  in if null acts
    then (if check then no-tac else all-tac) thm
    else (act-on ctxt (hd acts) THEN act ctxt false vs) thm end

fun cleanup ctxt = SUBGOAL (fn (t, i) => case Strengthen.goal-predicate t of
    st => resolve-tac ctxt [st-refl] i
  | - => all-tac)

fun tac ctxt = SUBGOAL (fn (t, -) => let
    val vs = get-vars t
  in if null vs then no-tac else ALLGOALS (intro-tac ctxt vs)
    THEN ALLGOALS (TRY o strg-proc ctxt)
    THEN act ctxt true (0 upto (length vs - 1))
    THEN ALLGOALS (cleanup ctxt)
    THEN Local-Defs.unfold-tac ctxt @{thms target-var-def}
  end)
```

fun both-tac ctxt = (*Datatype-Schematic.tac ctxt THEN'* (*TRY o tac ctxt*))
    ORELSE' tac ctxt

val method =
  Method.sections [*Datatype-Schematic.add-section*] >>
    (*fn* - => *fn ctxt* => *Method.SIMPLE-METHOD'* (*both-tac ctxt*));

end
⟩

method-setup *wpfix* = ⟨*WPFix.method*⟩

lemma *demo1*:
  (∃ *Ia Ib Ic Id Ra*.
    (*Ia* (*Suc 0*) ⟶ *Qa*)
  ∧ (*Ib* ⟶ *Qb*)
  ∧ (*Ic* ⟶ *Ra*)
  ∧ (*Id* ⟶ *Qc*)
  ∧ (*Id* ⟶ *Qd*)
  ∧ (*Qa* ∧ *Qb* ∧ *Qc* ∧ *Qd* ⟶ *Ia v* ∧ *Ib* ∧ *Ic* ∧ *Id*))
  apply (*intro exI conjI impI*)

  apply (*wpfix* | *assumption*)+
  apply *auto*
  done

lemma *demo2*:
  assumes *P*: ⋀*x*. *P* (*x* + *Suc x*) ⟶ *R* (*Inl x*)
        ⋀*x*. *P* ((*x* ∗ *2*) − *1*) ⟶ *R* (*Inr x*)
  assumes *P17*: *P 17*
  shows ∃ *I*. *I* (*Some 9*)
    ∧ (∀ *x*. *I x* ⟶ (*case x of None* ⇒ *R* (*Inl 8*) | *Some y* ⇒ *R* (*Inr y*)))
    ∧ (∀ *x*. *I x* ⟶ (*case x of None* ⇒ *R* (*Inr 9*) | *Some y* ⇒ *R* (*Inl* (*y* − *1*))))
  apply (*intro exI conjI*[*rotated*] *allI*)
    apply (*case-tac x*; *simp*)
     apply *wpfix*
     apply (*rule P*)
    apply *wpfix*
    apply (*rule P*)
   apply (*case-tac x*; *simp*)
    apply *wpfix*
    apply (*rule P*)
   apply *wpfix*
   apply (*rule P*)
  apply (*simp add*: *P17*)
  done

— Shows how to use *datatype-schematic* rules as "accessors".

lemma (**in** *datatype-schem-demo*) *demo3*:
  $\exists x. \forall a\ b.\ x\ (basic\ a\ b) = a$
  apply (*rule exI*, (*rule allI*)+)
  apply (*wpfix add*: *get-basic-0.simps*) — Only exposes ‘a‘ to the schematic.
  by (*rule refl*)

end

theory *WP*
imports
  WP-Pre
  WPFix
  ../../Apply-Debug
  ../../ml−helpers/MLUtils
begin

definition
  $triple\text{-}judgement :: ('a \Rightarrow bool) \Rightarrow 'b \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
where
 $triple\text{-}judgement\ pre\ body\ property = (\forall s.\ pre\ s \longrightarrow property\ s\ body)$

definition
  $postcondition :: ('r \Rightarrow 's \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow ('r \times 's)\ set)$
        $\Rightarrow 'a \Rightarrow 'b \Rightarrow bool$
where
 $postcondition\ P\ f = (\lambda a\ b.\ \forall (rv, s) \in f\ a\ b.\ P\ rv\ s)$

definition
  $postconditions :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool)$
where
 $postconditions\ P\ Q = (\lambda a\ b.\ P\ a\ b \wedge Q\ a\ b)$

lemma *conj-TrueI*: $P \Longrightarrow True \wedge P$ **by** *simp*
lemma *conj-TrueI2*: $P \Longrightarrow P \wedge True$ **by** *simp*

ML-file *WP−method.ML*

declare [[*wp-trace* = *false*]]

setup *WeakestPre.setup*

method-setup *wp* = ‹*WeakestPre.apply-wp-args*›
  applies weakest precondition rules

end

theory *WPC*

imports *WP-Pre*
keywords *wpc-setup* :: *thy-decl*

begin

definition
wpc-helper :: $(('a \Rightarrow bool) \times 'b\ set)$
$\Rightarrow (('a \Rightarrow bool) \times 'b\ set) \Rightarrow bool \Rightarrow bool$ **where**
wpc-helper $\equiv \lambda(P, P')\ (Q, Q')\ R.\ ((\forall s.\ P\ s \longrightarrow Q\ s) \wedge P' \subseteq Q') \longrightarrow R$

lemma *wpc-conj-process*:
$[\![$ *wpc-helper* $(P, P')\ (A, A')\ C$; *wpc-helper* $(P, P')\ (B, B')\ D\ ]\!]$
$\Longrightarrow$ *wpc-helper* $(P, P')\ (\lambda s.\ A\ s \wedge B\ s, A' \cap B')\ (C \wedge D)$
by (*clarsimp simp add*: *wpc-helper-def*)

lemma *wpc-all-process*:
$[\![\ \bigwedge x.$ *wpc-helper* $(P, P')\ (Q\ x, Q'\ x)\ (R\ x)\ ]\!]$
$\Longrightarrow$ *wpc-helper* $(P, P')\ (\lambda s.\ \forall x.\ Q\ x\ s, \{s.\ \forall x.\ s \in Q'\ x\})\ (\forall x.\ R\ x)$
by (*clarsimp simp*: *wpc-helper-def subset-iff*)

lemma *wpc-all-process-very-weak*:
$[\![\ \bigwedge x.$ *wpc-helper* $(P, P')\ (Q, Q')\ (R\ x)\ ]\!] \Longrightarrow$ *wpc-helper* $(P, P')\ (Q, Q')\ (\forall x.$
$R\ x)$
by (*clarsimp simp*: *wpc-helper-def*)

lemma *wpc-imp-process*:
$[\![\ Q \Longrightarrow$ *wpc-helper* $(P, P')\ (R, R')\ S\ ]\!]$
$\Longrightarrow$ *wpc-helper* $(P, P')\ (\lambda s.\ Q \longrightarrow R\ s, \{s.\ Q \longrightarrow s \in R'\})\ (Q \longrightarrow S)$
by (*clarsimp simp add*: *wpc-helper-def subset-iff*)

lemma *wpc-imp-process-weak*:
$[\![$ *wpc-helper* $(P, P')\ (R, R')\ S\ ]\!] \Longrightarrow$ *wpc-helper* $(P, P')\ (R, R')\ (Q \longrightarrow S)$
by (*clarsimp simp add*: *wpc-helper-def*)

lemmas *wpc-processors*
= *wpc-conj-process wpc-all-process wpc-imp-process*
lemmas *wpc-weak-processors*
= *wpc-conj-process wpc-all-process wpc-imp-process-weak*
lemmas *wpc-vweak-processors*
= *wpc-conj-process wpc-all-process-very-weak wpc-imp-process-weak*

lemma *wpc-helperI*:
wpc-helper $(P, P')\ (P, P')\ Q \Longrightarrow Q$
by (*simp add*: *wpc-helper-def*)

lemma *wpc-foo*: $[\![$ *undefined* x; *False* $]\!] \Longrightarrow P\ x$
by *simp*

lemma *foo*:

assumes *foo-elim*: $\bigwedge P\ Q\ h.\ [\![\ foo\ Q\ h;\ \bigwedge s.\ P\ s \Longrightarrow Q\ s\ ]\!] \Longrightarrow foo\ P\ h$
shows
$[\![\ \bigwedge x.\ foo\ (Q\ x)\ (f\ x);\ foo\ R\ g\ ]\!] \Longrightarrow$
$\quad foo\ (\lambda s.\ (\forall x.\ Q\ x\ s) \wedge (y = None \longrightarrow R\ s))$
$\quad\quad (case\ y\ of\ Some\ x \Rightarrow f\ x \mid None \Rightarrow g)$
by (*auto split*: *option.split intro*: *foo-elim*)

ML ‹

```
signature WPC = sig
  exception WPCFailed of string * term list * thm list;

  val foo-thm: thm;
  val iffd2-thm: thm;
  val wpc-helperI: thm;

  val instantiate-concl-pred: Proof.context -> cterm -> thm -> thm;

  val detect-term: Proof.context -> int -> thm -> cterm -> (cterm * term)
list;
  val detect-terms: Proof.context -> (term -> cterm -> thm -> int -> tactic)
-> int -> tactic;

  val split-term: thm list -> Proof.context -> term -> cterm -> thm -> int
-> tactic;

  val wp-cases-tac: thm list -> Proof.context -> int -> tactic;
  val wp-debug-tac: thm list -> Proof.context -> int -> tactic;
  val wp-cases-method: thm list -> (Proof.context -> Method.method) context-parser;

end;

structure WPCPredicateAndFinals = Theory-Data
(struct
    type T = (cterm * thm) list
    val empty = []
    val extend = I
    fun merge (xs, ys) =
        (* Order of predicates is important, so we can't reorder *)
        let val tms = map (Thm.term-of o fst) xs
            fun inxs x = exists (fn y => x aconv y) tms
            val ys' = filter (not o inxs o Thm.term-of o fst) ys
        in
            xs @ ys'
        end
end);

structure WeakestPreCases : WPC =
struct
```

```
exception WPCFailed of string * term list * thm list;

val iffd2-thm = @{thm iffD2};
val wpc-helperI = @{thm wpc-helperI};
val foo-thm = @{thm wpc-foo};

(* it looks like cterm-instantiate would do the job better,
   but this handles the case where ?'a must be instantiated
   to ?'a × ?'b *)
fun instantiate-concl-pred ctxt pred thm =
let
  val get-concl-pred  = (fst o strip-comb o HOLogic.dest-Trueprop o Thm.concl-of);
  val get-concl-predC = (Thm.cterm-of ctxt o get-concl-pred);

  val get-pred-tvar   = domain-type o Thm.typ-of o Thm.ctyp-of-cterm;
  val thm-pred        = get-concl-predC thm;
  val thm-pred-tvar   = Term.dest-TVar (get-pred-tvar thm-pred);
  val pred-tvar       = Thm.ctyp-of ctxt (get-pred-tvar pred);

  val thm2            = Thm.instantiate ([(thm-pred-tvar, pred-tvar)], []) thm;

  val thm2-pred       = Term.dest-Var (get-concl-pred thm2);
in
  Thm.instantiate ([], [(thm2-pred, pred)]) thm2
end;

fun detect-term ctxt n thm tm =
let
  val foo-thm-tm   = instantiate-concl-pred ctxt tm foo-thm;
  val matches      = resolve-tac ctxt [foo-thm-tm] n thm;
  val outcomes     = Seq.list-of matches;
  val get-goalterm = (HOLogic.dest-Trueprop o Logic.strip-assums-concl
                      o Envir.beta-eta-contract o hd o Thm.prems-of);
  val get-argument = hd o snd o strip-comb;
in
  map (pair tm o get-argument o get-goalterm) outcomes
end;

fun detect-terms ctxt tactic2 n thm =
let
  val pfs            = WPCPredicateAndFinals.get (Proof-Context.theory-of ctxt);
  val detects        = map (fn (tm, rl) => (detect-term ctxt n thm tm, rl)) pfs;
  val detects2       = filter (not o null o fst) detects;
  val ((pred, arg), fin)   = case detects2 of
                          [] => raise WPCFailed (detect-terms: no match, [], [thm])
                           | ((d3, fin) :: -) => (hd d3, fin)
in
  tactic2 arg pred fin n thm
```

```
end;

(* give each rule in the list one possible resolution outcome *)
fun resolve-each-once-tac ctxt thms i
    = fold (curry (APPEND'))
        (map (DETERM oo resolve-tac ctxt o single) thms)
        (K no-tac) i

fun resolve-single-tac ctxt rules n thm =
  case Seq.chop 2 (resolve-each-once-tac ctxt rules n thm)
  of ([], -) => raise WPCFailed
                      (resolve-single-tac: no rules could apply,
                       [], thm :: rules)
   | (- :: - :: -, -) => raise WPCFailed
                      (resolve-single-tac: multiple rules applied,
                       [], thm :: rules)
   | ([x], -) => Seq.single x;

fun split-term processors ctxt target pred fin =
let
  val hdTarget      = head-of target;
  val (constNm, -)  = dest-Const hdTarget handle TERM (-, tms)
                     => raise WPCFailed (split-term: couldn't dest-Const, tms, []);
  val split = case (Ctr-Sugar.ctr-sugar-of-case ctxt constNm) of
      SOME sugar => #split sugar
    | - => raise WPCFailed (split-term: not a case, [hdTarget], []);
  val subst         = split RS iffd2-thm;
  val subst2        = instantiate-concl-pred ctxt pred subst;
in
 (resolve-tac ctxt [subst2])
   THEN'
 (resolve-tac ctxt [wpc-helperI])
   THEN'
 (REPEAT-ALL-NEW (resolve-tac ctxt processors)
    THEN-ALL-NEW
  resolve-single-tac ctxt [fin])
end;

(* n.b. need to concretise the lazy sequence via a list to ensure exceptions
  have been raised already and catch them *)
fun wp-cases-tac processors ctxt n thm =
  detect-terms ctxt (split-term processors ctxt) n thm
      |> Seq.list-of |> Seq.of-list
    handle WPCFailed - => no-tac thm;

fun wp-debug-tac processors ctxt n thm =
  detect-terms ctxt (split-term processors ctxt) n thm
      |> Seq.list-of |> Seq.of-list
     handle WPCFailed e => (warning (@{make-string} (WPCFailed e)); no-tac
```

thm);

fun wp-cases-method processors = Scan.succeed (fn ctxt =>
  Method.SIMPLE-METHOD′ (wp-cases-tac processors ctxt));

local structure P = Parse and K = Keyword in

fun add-wpc tm thm lthy = let
  val ctxt = Local-Theory.target-of lthy
  val tm′ = (Syntax.read-term ctxt tm) |> Thm.cterm-of ctxt o Logic.varify-global
  val thm′ = Proof-Context.get-thm ctxt thm
in
  Local-Theory.background-theory (WPCPredicateAndFinals.map (fn xs => (tm′, thm′) :: xs)) lthy
end;

val - =
    Outer-Syntax.command
        @{command-keyword wpc-setup}
        Add wpc stuff
          (P.term −− P.name >> (fn (tm, thm) => Toplevel.local-theory NONE NONE (add-wpc tm thm)))

end;
end;

›

ML ‹

val wp-cases-tactic-weak = WeakestPreCases.wp-cases-tac @{thms wpc-weak-processors};
val wp-cases-method-strong = WeakestPreCases.wp-cases-method @{thms wpc-processors};
val wp-cases-method-weak = WeakestPreCases.wp-cases-method @{thms wpc-weak-processors};
val wp-cases-method-vweak = WeakestPreCases.wp-cases-method @{thms wpc-vweak-processors};

›

method-setup *wpc0* = ‹*wp-cases-method-strong*›
  case splitter for weakest−precondition proofs

method-setup *wpcw0* = ‹*wp-cases-method-weak*›
  weak−form case splitter for weakest−precondition proofs

method *wpc* = (*wp-pre*, *wpc0*)
method *wpcw* = (*wp-pre*, *wpcw0*)

definition
  wpc-test :: $'a$ *set* ⇒ ($'a \times 'b$) *set* ⇒ $'b$ *set* ⇒ *bool*
  where

wpc-test P R S ≡ (*R* `` *P*) ⊆ *S*

lemma *wpc-test-weaken*:
  ⟦ *wpc-test Q R S*; *P* ⊆ *Q* ⟧ ⟹ *wpc-test P R S*
  by (*simp add*: *wpc-test-def*, *blast*)

lemma *wpc-helper-validF*:
  wpc-test Q′ R S ⟹ *wpc-helper* (*P*, *P′*) (*Q*, *Q′*) (*wpc-test P′ R S*)
  by (*simp add*: *wpc-test-def wpc-helper-def*, *blast*)

setup ‹
let
  val tm = *Thm.cterm-of* @{*context*} (*Logic.varify-global* @{*term* λ*R*. *wpc-test P R S*});
  val thm = @{*thm wpc-helper-validF*};
in
  WPCPredicateAndFinals.map (*fn xs* => (*tm*, *thm*) :: *xs*)
end
›

lemma *set-conj-Int-simp*:
  {*s* ∈ *S*. *P s*} = *S* ∩ {*s*. *P s*}
  by *auto*

lemma *case-options-weak-wp*:
  ⟦ *wpc-test P R S*; ⋀*x*. *wpc-test P′* (*R′ x*) *S* ⟧
    ⟹ *wpc-test* (*P* ∩ *P′*) (*case opt of None* ⇒ *R* | *Some x* ⇒ *R′ x*) *S*
  apply (*rule wpc-test-weaken*)
   apply *wpcw*
    apply *assumption*
   apply *assumption*
  apply *simp*
  done

end

theory *Simp-No-Conditional*

imports *Main*

begin

Simplification without conditional rewriting. Setting the simplifier depth limit to zero prevents attempts at conditional rewriting. This should make the simplifier faster and more predictable on average. It may be particularly useful in derived tactics and methods to avoid situations where the simplifier repeatedly attempts and fails a conditional rewrite.

As always, there are caveats. Failing to perform a simple conditional rewrite may open the door to expensive alternatives. Various simprocs which are conditional in nature will not be deactivated.

ML ‹

```
structure Simp-No-Conditional = struct

val set-no-cond = Config.put Raw-Simplifier.simp-depth-limit 0

val simp-tac = Simplifier.simp-tac o set-no-cond
val asm-simp-tac = Simplifier.asm-simp-tac o set-no-cond
val full-simp-tac = Simplifier.full-simp-tac o set-no-cond
val asm-full-simp-tac = Simplifier.asm-full-simp-tac o set-no-cond

val clarsimp-tac = Clasimp.clarsimp-tac o set-no-cond
val auto-tac = Clasimp.auto-tac o set-no-cond

fun mk-method secs tac
  = Method.sections secs >> K (SIMPLE-METHOD' o tac)
val mk-clasimp-method = mk-method Clasimp.clasimp-modifiers

fun mk-clasimp-all-method tac =
  Method.sections Clasimp.clasimp-modifiers >> K (SIMPLE-METHOD o tac)

val simp-method = mk-method Simplifier.simp-modifiers
  (CHANGED-PROP oo asm-full-simp-tac)
val clarsimp-method = mk-clasimp-method (CHANGED-PROP oo clarsimp-tac)
val auto-method = mk-clasimp-all-method (CHANGED-PROP o auto-tac)

end
```

›

method-setup *simp-no-cond* = ‹*Simp-No-Conditional.simp-method*›
Simplification with no conditional simplification.

method-setup *clarsimp-no-cond* = ‹*Simp-No-Conditional.clarsimp-method*›
Clarsimp with no conditional simplification.

method-setup *auto-no-cond* = ‹*Simp-No-Conditional.auto-method*›
Auto with no conditional simplification.

end

theory *WPSimp*
imports
WP
WPC

WPFix
../../Simp-No-Conditional
begin

method *wpsimp* **uses** *wp wp-del simp simp-del split split-del cong comb comb-del*
=
((*determ* ‹*wpfix* | *wp add*: *wp del*: *wp-del comb*: *comb comb del*: *comb-del* | *wpc* |
clarsimp-no-cond simp: *simp simp del*: *simp-del split*: *split split del*: *split-del cong*: *cong* |
clarsimp simp: *simp simp del*: *simp-del split*: *split split del*: *split-del cong*: *cong*›)+)[*1*]

end

theory *NonDetMonadVCG*
imports
NonDetMonadLemmas
wp/WPSimp
Strengthen
begin

declare *K-def* [*simp*]

22 Satisfiability

The dual to validity: an existential instead of a universal quantifier for the post condition. In refinement, it is often sufficient to know that there is one state that satisfies a condition.

definition
exs-valid :: (*'a* ⇒ *bool*) ⇒ (*'a*, *'b*) *nondet-monad* ⇒
(*'b* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
(⦃-⦄ - ∃⦃-⦄)
where
exs-valid P f Q ≡ (∀ *s*. *P s* ⟶ (∃ (*rv*, *s'*) ∈ *fst* (*f s*). *Q rv s'*))

The above for the exception monad

definition
ex-exs-validE :: (*'a* ⇒ *bool*) ⇒ (*'a*, *'e* + *'b*) *nondet-monad* ⇒
(*'b* ⇒ *'a* ⇒ *bool*) ⇒ (*'e* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
(⦃-⦄ - ∃⦃-⦄, ⦃-⦄)
where
ex-exs-validE P f Q E ≡
exs-valid P f (λ*rv*. *case rv of Inl e* ⇒ *E e* | *Inr v* ⇒ *Q v*)

23 Lemmas

23.1 Determinism

lemma *det-set-iff*:
  $det\ f \Longrightarrow (r \in fst\ (f\ s)) = (fst\ (f\ s) = \{r\})$
  apply (*simp add*: *det-def*)
  apply (*rule iffI*)
  apply (*erule-tac x=s* **in** *allE*)
  apply *auto*
  done

lemma *return-det* [*iff*]:
  $det\ (return\ x)$
  by (*simp add*: *det-def return-def*)

lemma *put-det* [*iff*]:
  $det\ (put\ s)$
  by (*simp add*: *det-def put-def*)

lemma *get-det* [*iff*]:
  $det\ get$
  by (*simp add*: *det-def get-def*)

lemma *det-gets* [*iff*]:
  $det\ (gets\ f)$
  by (*auto simp add*: *gets-def det-def get-def return-def bind-def*)

lemma *det-UN*:
  $det\ f \Longrightarrow (\bigcup x \in fst\ (f\ s).\ g\ x) = (g\ (THE\ x.\ x \in fst\ (f\ s)))$
  unfolding *det-def*
  apply *simp*
  apply (*drule spec* [*of - s*])
  apply *clarsimp*
  done

lemma *bind-detI* [*simp*, *intro*!]:
  $[\![\ det\ f;\ \forall x.\ det\ (g\ x)\ ]\!] \Longrightarrow det\ (f >\!\!>\!\!= g)$
  apply (*simp add*: *bind-def det-def split-def*)
  apply *clarsimp*
  apply (*erule-tac x=s* **in** *allE*)
  apply *clarsimp*
  apply (*erule-tac x=a* **in** *allE*)
  apply (*erule-tac x=b* **in** *allE*)
  apply *clarsimp*
  done

lemma *the-run-stateI*:
  $fst\ (M\ s) = \{s'\} \Longrightarrow the\text{-}run\text{-}state\ M\ s = s'$
  by (*simp add*: *the-run-state-def*)

lemma *the-run-state-det*:
$[\![s' \in fst\ (M\ s);\ det\ M\ ]\!] \Longrightarrow the\text{-}run\text{-}state\ M\ s = s'$
by (*simp add*: *the-run-stateI det-set-iff*)

23.2 Lifting and Alternative Basic Definitions

lemma *liftE-liftM*: *liftE* = *liftM Inr*
apply (*rule ext*)
apply (*simp add*: *liftE-def liftM-def*)
done

lemma *liftME-liftM*: *liftME f* = *liftM* (*case-sum Inl* (*Inr* $\circ$ *f*))
apply (*rule ext*)
apply (*simp add*: *liftME-def liftM-def bindE-def returnOk-def lift-def*)
apply (*rule-tac f*=*bind x* **in** *arg-cong*)
apply (*rule ext*)
apply (*case-tac xa*)
apply (*simp-all add*: *lift-def throwError-def*)
done

lemma *liftE-bindE*:
(*liftE a*) >>=*E b* = *a* >>= *b*
apply (*simp add*: *liftE-def bindE-def lift-def bind-assoc*)
done

lemma *liftM-id*[*simp*]: *liftM id* = *id*
apply (*rule ext*)
apply (*simp add*: *liftM-def*)
done

lemma *liftM-bind*:
(*liftM t f* >>= *g*) = (*f* >>= (λx. *g* (*t x*)))
by (*simp add*: *liftM-def bind-assoc*)

lemma *gets-bind-ign*: *gets f* >>= (λx. *m*) = *m*
apply (*rule ext*)
apply (*simp add*: *bind-def simpler-gets-def*)
done

lemma *get-bind-apply*: (*get* >>= *f*) *x* = *f x x*
by (*simp add*: *get-def bind-def*)

lemma *exec-gets*:
(*gets f* >>= *m*) *s* = *m* (*f s*) *s*
by (*simp add*: *simpler-gets-def bind-def*)

lemma *exec-get*:
(*get* >>= *m*) *s* = *m s s*

by (*simp add*: *get-def bind-def*)

lemma *bind-eqI*:
$[\![f = f'; \bigwedge x.\ g\ x = g'\ x]\!] \Longrightarrow f >>= g = f' >>= g'$
apply (*rule ext*)
apply (*simp add*: *bind-def*)
apply (*auto simp*: *split-def*)
done

23.3 Simplification Rules for Lifted And/Or

lemma *pred-andE*[*elim*!]: $[\![(A\ and\ B)\ x; [\![A\ x; B\ x]\!] \Longrightarrow R]\!] \Longrightarrow R$
by(*simp add*:*pred-conj-def*)

lemma *pred-andI*[*intro*!]: $[\![A\ x; B\ x]\!] \Longrightarrow (A\ and\ B)\ x$
by(*simp add*:*pred-conj-def*)

lemma *pred-conj-app*[*simp*]: $(P\ and\ Q)\ x = (P\ x \wedge Q\ x)$
by(*simp add*:*pred-conj-def*)

lemma *bipred-andE*[*elim*!]: $[\![(A\ And\ B)\ x\ y; [\![A\ x\ y; B\ x\ y]\!] \Longrightarrow R]\!] \Longrightarrow R$
by(*simp add*:*bipred-conj-def*)

lemma *bipred-andI*[*intro*!]: $[\![A\ x\ y; B\ x\ y]\!] \Longrightarrow (A\ And\ B)\ x\ y$
by (*simp add*:*bipred-conj-def*)

lemma *bipred-conj-app*[*simp*]: $(P\ And\ Q)\ x = (P\ x\ and\ Q\ x)$
by(*simp add*:*pred-conj-def bipred-conj-def*)

lemma *pred-disjE*[*elim*!]: $[\![(P\ or\ Q)\ x; P\ x \Longrightarrow R; Q\ x \Longrightarrow R]\!] \Longrightarrow R$
by (*fastforce simp*: *pred-disj-def*)

lemma *pred-disjI1*[*intro*]: $P\ x \Longrightarrow (P\ or\ Q)\ x$
by (*simp add*: *pred-disj-def*)

lemma *pred-disjI2*[*intro*]: $Q\ x \Longrightarrow (P\ or\ Q)\ x$
by (*simp add*: *pred-disj-def*)

lemma *pred-disj-app*[*simp*]: $(P\ or\ Q)\ x = (P\ x \vee Q\ x)$
by *auto*

lemma *bipred-disjI1*[*intro*]: $P\ x\ y \Longrightarrow (P\ Or\ Q)\ x\ y$
by (*simp add*: *bipred-disj-def*)

lemma *bipred-disjI2*[*intro*]: $Q\ x\ y \Longrightarrow (P\ Or\ Q)\ x\ y$
by (*simp add*: *bipred-disj-def*)

lemma *bipred-disj-app*[*simp*]: $(P\ Or\ Q)\ x = (P\ x\ or\ Q\ x)$
by(*simp add*:*pred-disj-def bipred-disj-def*)

lemma *pred-notnotD*[*simp*]: $(not\ not\ P) = P$
by(*simp add*:*pred-neg-def*)

lemma *pred-and-true*[*simp*]: $(P\ and\ \top) = P$
by(*simp add*:*pred-conj-def*)

lemma *pred-and-true-var*[*simp*]: $(\top\ and\ P) = P$
by(*simp add*:*pred-conj-def*)

lemma *pred-and-false*[*simp*]: $(P\ and\ \bot) = \bot$
by(*simp add*:*pred-conj-def*)

lemma *pred-and-false-var*[*simp*]: $(\bot\ and\ P) = \bot$
by(*simp add*:*pred-conj-def*)

lemma *pred-conj-assoc*:
$(P\ and\ Q\ and\ R) = (P\ and\ (Q\ and\ R))$
unfolding *pred-conj-def* **by** *simp*

23.4 Hoare Logic Rules

lemma *validE-def2*:
$\{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|R|\!\} \equiv \forall s.\ P\ s \longrightarrow (\forall (r,s') \in fst\ (f\ s).\ case\ r\ of\ Inr\ b \Rightarrow Q\ b\ s'$
$\quad | \ Inl\ a \Rightarrow R\ a\ s')$
by (*unfold valid-def validE-def*)

lemma *seq'*:
$[\![\ \{\!|A|\!\}\ f\ \{\!|B|\!\};$
$\quad \forall x.\ P\ x \longrightarrow \{\!|C|\!\}\ g\ x\ \{\!|D|\!\};$
$\quad \forall x\ s.\ B\ x\ s \longrightarrow P\ x \wedge C\ s\ ]\!] \Longrightarrow$
$\{\!|A|\!\}\ do\ x \leftarrow f;\ g\ x\ od\ \{\!|D|\!\}$
apply (*clarsimp simp*: *valid-def bind-def*)
apply *fastforce*
done

lemma *seq*:
assumes *f-valid*: $\{\!|A|\!\}\ f\ \{\!|B|\!\}$
assumes *g-valid*: $\bigwedge x.\ P\ x \Longrightarrow \{\!|C|\!\}\ g\ x\ \{\!|D|\!\}$
assumes *bind*: $\bigwedge x\ s.\ B\ x\ s \Longrightarrow P\ x \wedge C\ s$
shows $\{\!|A|\!\}\ do\ x \leftarrow f;\ g\ x\ od\ \{\!|D|\!\}$
apply (*insert f-valid g-valid bind*)
apply (*blast intro*: *seq'*)
done

lemma *seq-ext'*:
$[\![\ \{\!|A|\!\}\ f\ \{\!|B|\!\};$
$\quad \forall x.\ \{\!|B\ x|\!\}\ g\ x\ \{\!|C|\!\}\ ]\!] \Longrightarrow$
$\{\!|A|\!\}\ do\ x \leftarrow f;\ g\ x\ od\ \{\!|C|\!\}$

by (*fastforce simp*: *valid-def bind-def Let-def split-def*)

lemma *seq-ext*:
assumes *f-valid*: $\{\!|A|\!\}$ f $\{\!|B|\!\}$
assumes *g-valid*: $\bigwedge x.$ $\{\!|B\ x|\!\}$ $g\ x$ $\{\!|C|\!\}$
shows $\{\!|A|\!\}$ *do* $x \leftarrow f;$ $g\ x$ *od* $\{\!|C|\!\}$
apply(*insert f-valid g-valid*)
apply(*blast intro*: *seq-ext'*)
done

lemma *seqE'*:
$[\![$ $\{\!|A|\!\}$ f $\{\!|B|\!\},\{\!|E|\!\};$
$\forall x.$ $\{\!|B\ x|\!\}$ $g\ x$ $\{\!|C|\!\},\{\!|E|\!\}$ $]\!] \Longrightarrow$
$\{\!|A|\!\}$ *doE* $x \leftarrow f;$ $g\ x$ *odE* $\{\!|C|\!\},\{\!|E|\!\}$
apply(*simp add*:*bindE-def lift-def bind-def Let-def split-def*)
apply(*clarsimp simp*:*validE-def2*)
apply (*fastforce simp add*: *throwError-def return-def lift-def*
split: *sum.splits*)
done

lemma *seqE*:
assumes *f-valid*: $\{\!|A|\!\}$ f $\{\!|B|\!\},\{\!|E|\!\}$
assumes *g-valid*: $\bigwedge x.$ $\{\!|B\ x|\!\}$ $g\ x$ $\{\!|C|\!\},\{\!|E|\!\}$
shows $\{\!|A|\!\}$ *doE* $x \leftarrow f;$ $g\ x$ *odE* $\{\!|C|\!\},\{\!|E|\!\}$
apply(*insert f-valid g-valid*)
apply(*blast intro*: *seqE'*)
done

lemma *hoare-TrueI*: $\{\!|P|\!\}$ f $\{\!|\lambda\text{-}.\ \top|\!\}$
by (*simp add*: *valid-def*)

lemma *hoareE-TrueI*: $\{\!|P|\!\}$ f $\{\!|\lambda\text{-}.\ \top|\!\},$ $\{\!|\lambda r.\ \top|\!\}$
by (*simp add*: *validE-def valid-def*)

lemma *hoare-True-E-R* [*simp*]:
$\{\!|P|\!\}$ f $\{\!|\lambda r\ s.\ True|\!\},$ $-$
by (*auto simp add*: *validE-R-def validE-def valid-def split*: *sum.splits*)

lemma *hoare-post-conj* [*intro*]:
$[\![$ $\{\!|$ P $|\!\}$ a $\{\!|$ Q $|\!\};$ $\{\!|$ P $|\!\}$ a $\{\!|$ R $|\!\}$ $]\!] \Longrightarrow$ $\{\!|$ P $|\!\}$ a $\{\!|$ Q *And* R $|\!\}$
by (*fastforce simp*: *valid-def split-def bipred-conj-def*)

lemma *hoare-pre-disj* [*intro*]:
$[\![$ $\{\!|$ P $|\!\}$ a $\{\!|$ R $|\!\};$ $\{\!|$ Q $|\!\}$ a $\{\!|$ R $|\!\}$ $]\!] \Longrightarrow$ $\{\!|$ P *or* Q $|\!\}$ a $\{\!|$ R $|\!\}$
by (*simp add*:*valid-def pred-disj-def*)

lemma *hoare-conj*:
$[\![$ $\{\!|P|\!\}$ f $\{\!|Q|\!\};$ $\{\!|P'|\!\}$ f $\{\!|Q'|\!\}$ $]\!] \Longrightarrow$ $\{\!|P$ *and* $P'|\!\}$ f $\{\!|Q$ *And* $Q'|\!\}$
unfolding *valid-def* **by** *auto*

lemma *hoare-post-taut*: $\{\!\!\{\ P\ \}\!\!\}\ a\ \{\!\!\{\ \top\top\ \}\!\!\}$
by (*simp add*:*valid-def*)

lemma *wp-post-taut*: $\{\!\!\{\lambda r.\ True\}\!\!\}\ f\ \{\!\!\{\lambda r\ s.\ True\}\!\!\}$
by (*rule hoare-post-taut*)

lemma *wp-post-tautE*: $\{\!\!\{\lambda r.\ True\}\!\!\}\ f\ \{\!\!\{\lambda r\ s.\ True\}\!\!\},\{\!\!\{\lambda f\ s.\ True\}\!\!\}$
proof −
have P: $\bigwedge r.$ (*case r of Inl a* $\Rightarrow$ *True* | - $\Rightarrow$ *True*) = *True*
by (*case-tac r*, *simp-all*)
show *?thesis*
by (*simp add*: *validE-def P wp-post-taut*)
qed

lemma *hoare-pre-cont* [*simp*]: $\{\!\!\{\ \bot\ \}\!\!\}\ a\ \{\!\!\{\ P\ \}\!\!\}$
by (*simp add*:*valid-def*)

23.5 Strongest Postcondition Rules

lemma *get-sp*:
$\{\!\!\{P\}\!\!\}\ get\ \{\!\!\{\lambda a\ s.\ s = a \wedge P\ s\}\!\!\}$
by(*simp add*:*get-def valid-def*)

lemma *put-sp*:
$\{\!\!\{\top\}\!\!\}\ put\ a\ \{\!\!\{\lambda\text{-}\ s.\ s = a\}\!\!\}$
by(*simp add*:*put-def valid-def*)

lemma *return-sp*:
$\{\!\!\{P\}\!\!\}\ return\ a\ \{\!\!\{\lambda b\ s.\ b = a \wedge P\ s\}\!\!\}$
by(*simp add*:*return-def valid-def*)

lemma *assert-sp*:
$\{\!\!\{\ P\ \}\!\!\}\ assert\ Q\ \{\!\!\{\ \lambda r\ s.\ P\ s \wedge Q\ \}\!\!\}$
by (*simp add*: *assert-def fail-def return-def valid-def*)

lemma *hoare-gets-sp*:
$\{\!\!\{P\}\!\!\}\ gets\ f\ \{\!\!\{\lambda rv\ s.\ rv = f\ s \wedge P\ s\}\!\!\}$
by (*simp add*: *valid-def simpler-gets-def*)

lemma *hoare-return-drop-var* [*iff*]: $\{\!\!\{\ Q\ \}\!\!\}\ return\ x\ \{\!\!\{\ \lambda r.\ Q\ \}\!\!\}$
by (*simp add*:*valid-def return-def*)

lemma *hoare-gets* [*intro*]: $[\![\ \bigwedge s.\ P\ s \Longrightarrow Q\ (f\ s)\ s\ ]\!] \Longrightarrow \{\!\!\{\ P\ \}\!\!\}\ gets\ f\ \{\!\!\{\ Q\ \}\!\!\}$
by (*simp add*:*valid-def gets-def get-def bind-def return-def*)

lemma *hoare-modifyE-var*:
$[\![\ \bigwedge s.\ P\ s \Longrightarrow Q\ (f\ s)\ ]\!] \Longrightarrow \{\!\!\{\ P\ \}\!\!\}\ modify\ f\ \{\!\!\{\ \lambda r\ s.\ Q\ s\ \}\!\!\}$
by(*simp add*: *valid-def modify-def put-def get-def bind-def*)

lemma *hoare-if*:
$\llbracket$ $P \Longrightarrow \{\!|\ Q\ |\!\}\ a\ \{\!|\ R\ |\!\}$; $\neg\ P \Longrightarrow \{\!|\ Q\ |\!\}\ b\ \{\!|\ R\ |\!\}$ $\rrbracket \Longrightarrow$
$\{\!|\ Q\ |\!\}$ *if P then a else b* $\{\!|\ R\ |\!\}$
by (*simp add*:*valid-def*)

lemma *hoare-pre-subst*: $\llbracket$ $A = B$; $\{\!|A|\!\}\ a\ \{\!|C|\!\}$ $\rrbracket \Longrightarrow \{\!|B|\!\}\ a\ \{\!|C|\!\}$
by(*clarsimp simp*:*valid-def split-def*)

lemma *hoare-post-subst*: $\llbracket$ $B = C$; $\{\!|A|\!\}\ a\ \{\!|B|\!\}$ $\rrbracket \Longrightarrow \{\!|A|\!\}\ a\ \{\!|C|\!\}$
by(*clarsimp simp*:*valid-def split-def*)

lemma *hoare-pre-tautI*: $\llbracket$ $\{\!|A\ \mathit{and}\ P|\!\}\ a\ \{\!|B|\!\}$; $\{\!|A\ \mathit{and\ not}\ P|\!\}\ a\ \{\!|B|\!\}$ $\rrbracket \Longrightarrow \{\!|A|\!\}\ a$
$\{\!|B|\!\}$
by(*fastforce simp*:*valid-def split-def pred-conj-def pred-neg-def*)

lemma *hoare-pre-imp*: $\llbracket$ $\bigwedge s.\ P\ s \Longrightarrow Q\ s$; $\{\!|Q|\!\}\ a\ \{\!|R|\!\}$ $\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|R|\!\}$
by (*fastforce simp add*:*valid-def*)

lemma *hoare-post-imp*: $\llbracket$ $\bigwedge r\ s.\ Q\ r\ s \Longrightarrow R\ r\ s$; $\{\!|P|\!\}\ a\ \{\!|Q|\!\}$ $\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|R|\!\}$
by(*fastforce simp*:*valid-def split-def*)

lemma *hoare-post-impErr'*: $\llbracket$ $\{\!|P|\!\}\ a\ \{\!|Q|\!\},\{\!|E|\!\}$;
$\forall\, r\ s.\ Q\ r\ s \longrightarrow R\ r\ s$;
$\forall\, e\ s.\ E\ e\ s \longrightarrow F\ e\ s$ $\rrbracket \Longrightarrow$
$\{\!|P|\!\}\ a\ \{\!|R|\!\},\{\!|F|\!\}$
apply (*simp add*: *validE-def*)
apply (*rule-tac Q*=$\lambda r\ s.$ *case r of Inl a* $\Rightarrow$ *E a s* | *Inr b* $\Rightarrow$ *Q b s* **in** *hoare-post-imp*)
apply (*case-tac r*)
apply *simp-all*
done

lemma *hoare-post-impErr*: $\llbracket$ $\{\!|P|\!\}\ a\ \{\!|Q|\!\},\{\!|E|\!\}$;
$\bigwedge r\ s.\ Q\ r\ s \Longrightarrow R\ r\ s$;
$\bigwedge e\ s.\ E\ e\ s \Longrightarrow F\ e\ s$ $\rrbracket \Longrightarrow$
$\{\!|P|\!\}\ a\ \{\!|R|\!\},\{\!|F|\!\}$
apply (*blast intro*: *hoare-post-impErr'*)
done

lemma *hoare-validE-cases*:
$\llbracket$ $\{\!|\ P\ |\!\}\ f\ \{\!|\ Q\ |\!\},\ \{\!|\ \lambda\text{-}\ \text{-}.\ \mathit{True}\ |\!\}$; $\{\!|\ P\ |\!\}\ f\ \{\!|\ \lambda\text{-}\ \text{-}.\ \mathit{True}\ |\!\},\ \{\!|\ R\ |\!\}$ $\rrbracket$
$\Longrightarrow \{\!|\ P\ |\!\}\ f\ \{\!|\ Q\ |\!\},\ \{\!|\ R\ |\!\}$
by (*simp add*: *validE-def valid-def split*: *sum.splits*) *blast*

lemma *hoare-post-imp-dc*:
$\llbracket\{\!|P|\!\}\ a\ \{\!|\lambda r.\ Q|\!\}$; $\bigwedge s.\ Q\ s \Longrightarrow R\ s\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|\lambda r.\ R|\!\},\{\!|\lambda r.\ R|\!\}$
by (*simp add*: *validE-def valid-def split*: *sum.splits*) *blast*

lemma *hoare-post-imp-dc2*:

$\llbracket\{\!|P|\!\}\ a\ \{\!|\lambda r.\ Q|\!\};\ \bigwedge s.\ Q\ s \Longrightarrow R\ s\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|\lambda r.\ R|\!\},\{\!|\lambda r\ s.\ True|\!\}$
by (*simp add*: *validE-def valid-def split*: *sum.splits*) *blast*

lemma *hoare-post-imp-dc2E*:
$\llbracket\{\!|P|\!\}\ a\ \{\!|\lambda r.\ Q|\!\};\ \bigwedge s.\ Q\ s \Longrightarrow R\ s\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|\lambda r\ s.\ True|\!\},\ \{\!|\lambda r.\ R|\!\}$
by (*simp add*: *validE-def valid-def split*: *sum.splits*) *fast*

lemma *hoare-post-imp-dc2E-actual*:
$\llbracket\{\!|P|\!\}\ a\ \{\!|\lambda r.\ R|\!\}\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|\lambda r\ s.\ True|\!\},\ \{\!|\lambda r.\ R|\!\}$
by (*simp add*: *validE-def valid-def split*: *sum.splits*) *fast*

lemma *hoare-post-imp-dc2-actual*:
$\llbracket\{\!|P|\!\}\ a\ \{\!|\lambda r.\ R|\!\}\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|\lambda r.\ R|\!\},\ \{\!|\lambda r\ s.\ True|\!\}$
by (*simp add*: *validE-def valid-def split*: *sum.splits*) *fast*

lemma *hoare-post-impE*: $\llbracket\ \bigwedge r\ s.\ Q\ r\ s \Longrightarrow R\ r\ s;\ \{\!|P|\!\}\ a\ \{\!|Q|\!\}\ \rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|R|\!\}$
by (*fastforce simp*:*valid-def split-def*)

lemma *hoare-conjD1*:
$\{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q\ rv\ and\ R\ rv|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q\ rv|\!\}$
unfolding *valid-def* **by** *auto*

lemma *hoare-conjD2*:
$\{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q\ rv\ and\ R\ rv|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda rv.\ R\ rv|\!\}$
unfolding *valid-def* **by** *auto*

lemma *hoare-post-disjI1*:
$\{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q\ rv|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q\ rv\ or\ R\ rv|\!\}$
unfolding *valid-def* **by** *auto*

lemma *hoare-post-disjI2*:
$\{\!|P|\!\}\ f\ \{\!|\lambda rv.\ R\ rv|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q\ rv\ or\ R\ rv|\!\}$
unfolding *valid-def* **by** *auto*

lemma *hoare-weaken-pre*:
$\llbracket\{\!|Q|\!\}\ a\ \{\!|R|\!\};\ \bigwedge s.\ P\ s \Longrightarrow Q\ s\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|R|\!\}$
apply (*rule hoare-pre-imp*)
 prefer *2*
 apply *assumption*
apply *blast*
done

lemma *hoare-strengthen-post*:
$\llbracket\{\!|P|\!\}\ a\ \{\!|Q|\!\};\ \bigwedge r\ s.\ Q\ r\ s \Longrightarrow R\ r\ s\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \{\!|R|\!\}$
apply (*rule hoare-post-imp*)
 prefer *2*
 apply *assumption*
apply *blast*
done

lemma *use-valid*: $\llbracket(r,\ s') \in fst\ (f\ s);\ \{\!|P|\!\}\ f\ \{\!|Q|\!\};\ P\ s\ \rrbracket \Longrightarrow Q\ r\ s'$
  apply (*simp add*: *valid-def*)
  apply *blast*
  done

lemma *use-validE-norm*: $\llbracket\ (Inr\ r',\ s') \in fst\ (B\ s);\ \{\!|\ P\ |\!\}\ B\ \{\!|\ Q\ |\!\},\{\!|\ E\ |\!\};\ P\ s\ \rrbracket \Longrightarrow Q\ r'\ s'$
  apply (*clarsimp simp*: *validE-def valid-def*)
  apply *force*
  done

lemma *use-validE-except*: $\llbracket\ (Inl\ r',\ s') \in fst\ (B\ s);\ \{\!|\ P\ |\!\}\ B\ \{\!|\ Q\ |\!\},\{\!|\ E\ |\!\};\ P\ s\ \rrbracket \Longrightarrow E\ r'\ s'$
  apply (*clarsimp simp*: *validE-def valid-def*)
  apply *force*
  done

lemma *in-inv-by-hoareD*:
  $\llbracket\ \bigwedge P.\ \{\!|P|\!\}\ f\ \{\!|\lambda\text{-}.\ P|\!\};\ (x,s') \in fst\ (f\ s)\ \rrbracket \Longrightarrow s' = s$
  by (*auto simp add*: *valid-def*) *blast*

23.6 Satisfiability

lemma *exs-hoare-post-imp*: $\llbracket\bigwedge r\ s.\ Q\ r\ s \Longrightarrow R\ r\ s;\ \{\!|P|\!\}\ a\ \exists\{\!|Q|\!\}\rrbracket \Longrightarrow \{\!|P|\!\}\ a\ \exists\{\!|R|\!\}$
  apply (*simp add*: *exs-valid-def*)
  apply *safe*
  apply (*erule-tac x=s* **in** *allE*, *simp*)
  apply *blast*
  done

lemma *use-exs-valid*: $\llbracket\{\!|P|\!\}\ f\ \exists\{\!|Q|\!\};\ P\ s\ \rrbracket \Longrightarrow \exists(r,\ s') \in fst\ (f\ s).\ Q\ r\ s'$
  by (*simp add*: *exs-valid-def*)

definition *exs-postcondition* $P\ f \equiv (\lambda a\ b.\ \exists(rv,\ s)\in f\ a\ b.\ P\ rv\ s)$

lemma *exs-valid-is-triple*:
  exs-valid $P\ f\ Q$ = *triple-judgement* $P\ f$ (*exs-postcondition* Q $(\lambda s\ f.\ fst\ (f\ s)))$
  by (*simp add*: *triple-judgement-def exs-postcondition-def exs-valid-def*)

lemmas [*wp-trip*] = *exs-valid-is-triple*

lemma *exs-valid-weaken-pre*[*wp-pre*]:
  $\llbracket\ \{\!|\ P'\ |\!\}\ f\ \exists\{\!|\ Q\ |\!\};\ \bigwedge s.\ P\ s \Longrightarrow P'\ s\ \rrbracket \Longrightarrow \{\!|\ P\ |\!\}\ f\ \exists\{\!|\ Q\ |\!\}$
  apply *atomize*
  apply (*clarsimp simp*: *exs-valid-def*)
  done

lemma *exs-valid-chain*:
$[\![\; \{\!\mid P \mid\!\} \; f \; \exists\{\!\mid Q \mid\!\}; \bigwedge s. \; R \; s \Longrightarrow P \; s; \bigwedge r \; s. \; Q \; r \; s \Longrightarrow S \; r \; s \;]\!] \Longrightarrow \{\!\mid R \mid\!\} \; f \; \exists\{\!\mid S \mid\!\}$
apply *atomize*
apply (*fastforce simp*: *exs-valid-def Bex-def*)
done

lemma *exs-valid-assume-pre*:
$[\![\; \bigwedge s. \; P \; s \Longrightarrow \{\!\mid P \mid\!\} \; f \; \exists\{\!\mid Q \mid\!\} \;]\!] \Longrightarrow \{\!\mid P \mid\!\} \; f \; \exists\{\!\mid Q \mid\!\}$
apply (*fastforce simp*: *exs-valid-def*)
done

lemma *exs-valid-bind* [*wp-split*]:
$[\![\; \bigwedge x. \; \{\!\mid B \; x \mid\!\} \; g \; x \; \exists\{\!\mid C \mid\!\}; \{\!\mid A \mid\!\} \; f \; \exists\{\!\mid B \mid\!\} \;]\!] \Longrightarrow \{\!\mid A \mid\!\} \; f >>= (\lambda x. \; g \; x) \; \exists\{\!\mid C \mid\!\}$
apply *atomize*
apply (*clarsimp simp*: *exs-valid-def bind-def′*)
apply *blast*
done

lemma *exs-valid-return* [*wp*]:
$\{\!\mid Q \; v \mid\!\} \; return \; v \; \exists\{\!\mid Q \mid\!\}$
by (*clarsimp simp*: *exs-valid-def return-def*)

lemma *exs-valid-select* [*wp*]:
$\{\!\mid \lambda s. \; \exists \, r \in S. \; Q \; r \; s \mid\!\} \; select \; S \; \exists\{\!\mid Q \mid\!\}$
by (*clarsimp simp*: *exs-valid-def select-def*)

lemma *exs-valid-get* [*wp*]:
$\{\!\mid \lambda s. \; Q \; s \; s \mid\!\} \; get \; \exists\{\!\mid Q \mid\!\}$
by (*clarsimp simp*: *exs-valid-def get-def*)

lemma *exs-valid-gets* [*wp*]:
$\{\!\mid \lambda s. \; Q \; (f \; s) \; s \mid\!\} \; gets \; f \; \exists\{\!\mid Q \mid\!\}$
by (*clarsimp simp*: *gets-def*) *wp*

lemma *exs-valid-put* [*wp*]:
$\{\!\mid Q \; v \mid\!\} \; put \; v \; \exists\{\!\mid Q \mid\!\}$
by (*clarsimp simp*: *put-def exs-valid-def*)

lemma *exs-valid-state-assert* [*wp*]:
$\{\!\mid \lambda s. \; Q \; () \; s \wedge G \; s \mid\!\} \; state\text{-}assert \; G \; \exists\{\!\mid Q \mid\!\}$
by (*clarsimp simp*: *state-assert-def exs-valid-def get-def*
assert-def bind-def′ return-def)

lemmas *exs-valid-guard* = *exs-valid-state-assert*

lemma *exs-valid-fail* [*wp*]:
$\{\!\mid \lambda\text{-}. \; False \mid\!\} \; fail \; \exists\{\!\mid Q \mid\!\}$
by (*clarsimp simp*: *fail-def exs-valid-def*)

lemma *exs-valid-condition* [*wp*]:
$[\![\ \{\!|\ P\ |\!\}\ L\ \exists\{\!|\ Q\ |\!\};\ \{\!|\ P'\ |\!\}\ R\ \exists\{\!|\ Q\ |\!\}\ ]\!] \Longrightarrow$
$\{\!|\ \lambda s.\ (C\ s \wedge P\ s) \vee (\neg\ C\ s \wedge P'\ s)\ |\!\}\ \mathit{condition}\ C\ L\ R\ \exists\{\!|\ Q\ |\!\}$
by (*clarsimp simp*: *condition-def exs-valid-def split*: *sum.splits*)

23.7 MISC

lemma *hoare-return-simp*:
$\{\!|P|\!\}\ \mathit{return}\ x\ \{\!|Q|\!\} = (\forall s.\ P\ s \longrightarrow Q\ x\ s)$
by (*simp add*: *valid-def return-def*)

lemma *hoare-gen-asm*:
$(P \Longrightarrow \{\!|P'|\!\}\ f\ \{\!|Q|\!\}) \Longrightarrow \{\!|P'\ \mathit{and}\ K\ P|\!\}\ f\ \{\!|Q|\!\}$
by (*fastforce simp add*: *valid-def*)

lemma *hoare-gen-asm-lk*:
$(P \Longrightarrow \{\!|P'|\!\}\ f\ \{\!|Q|\!\}) \Longrightarrow \{\!|K\ P\ \mathit{and}\ P'|\!\}\ f\ \{\!|Q|\!\}$
by (*fastforce simp add*: *valid-def*)

— Useful for forward reasoning, when P is known. The first version allows weakening the precondition.
lemma *hoare-gen-asm-spec'*:
$(\bigwedge s.\ P\ s \Longrightarrow S \wedge R\ s)$
$\Longrightarrow (S \Longrightarrow \{\!|R|\!\}\ f\ \{\!|Q|\!\})$
$\Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\}$
by (*fastforce simp*: *valid-def*)

lemma *hoare-gen-asm-spec*:
$(\bigwedge s.\ P\ s \Longrightarrow S)$
$\Longrightarrow (S \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\})$
$\Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\}$
by (*rule hoare-gen-asm-spec'*[**where** $S=S$ **and** $R=P$]) *simp*

lemma *hoare-conjI*:
$[\![\ \{\!|P|\!\}\ f\ \{\!|Q|\!\};\ \{\!|P|\!\}\ f\ \{\!|R|\!\}\ ]\!] \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ r\ s \wedge R\ r\ s|\!\}$
unfolding *valid-def* **by** *blast*

lemma *hoare-disjI1*:
$[\![\ \{\!|P|\!\}\ f\ \{\!|Q|\!\}\ ]\!] \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ r\ s \vee\ R\ r\ s\ |\!\}$
unfolding *valid-def* **by** *blast*

lemma *hoare-disjI2*:
$[\![\ \{\!|P|\!\}\ f\ \{\!|R|\!\}\ ]\!] \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ r\ s \vee\ R\ r\ s\ |\!\}$
unfolding *valid-def* **by** *blast*

lemma *hoare-assume-pre*:
$(\bigwedge s.\ P\ s \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\}) \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\}$
by (*auto simp*: *valid-def*)

lemma *hoare-returnOk-sp*:
$\{\!|P|\!\}$ *returnOk* x $\{\!|\lambda r\ s.\ r = x \wedge P\ s|\!\}, \{\!|Q|\!\}$
by (*simp add*: *valid-def validE-def returnOk-def return-def*)

lemma *hoare-assume-preE*:
$(\bigwedge s.\ P\ s \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|R|\!\}) \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|R|\!\}$
by (*auto simp*: *valid-def validE-def*)

lemma *hoare-allI*:
$(\bigwedge x.\ \{\!|P|\!\}f\{\!|Q\ x|\!\}) \Longrightarrow \{\!|P|\!\}f\{\!|\lambda r\ s.\ \forall x.\ Q\ x\ r\ s|\!\}$
by (*simp add*: *valid-def*) *blast*

lemma *validE-allI*:
$(\bigwedge x.\ \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ x\ r\ s|\!\},\{\!|E|\!\}) \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ \forall x.\ Q\ x\ r\ s|\!\},\{\!|E|\!\}$
by (*fastforce simp*: *valid-def validE-def split*: *sum.splits*)

lemma *hoare-exI*:
$\{\!|P|\!\}\ f\ \{\!|Q\ x|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ \exists x.\ Q\ x\ r\ s|\!\}$
by (*simp add*: *valid-def*) *blast*

lemma *hoare-impI*:
$(R \Longrightarrow \{\!|P|\!\}f\{\!|Q|\!\}) \Longrightarrow \{\!|P|\!\}f\{\!|\lambda r\ s.\ R \longrightarrow Q\ r\ s|\!\}$
by (*simp add*: *valid-def*) *blast*

lemma *validE-impI*:
$[\![\bigwedge E.\ \{\!|P|\!\}\ f\ \{\!|\lambda\text{- -}.\ True|\!\},\{\!|E|\!\};\ (P' \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\})]\!] \Longrightarrow$
$\{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ P' \longrightarrow Q\ r\ s|\!\}, \{\!|E|\!\}$
by (*fastforce simp*: *validE-def valid-def split*: *sum.splits*)

lemma *hoare-case-option-wp*:
$[\![\ \{\!|P|\!\}\ f\ None\ \{\!|Q|\!\};$
$\bigwedge x.\ \ \{\!|P'\ x|\!\}\ f\ (Some\ x)\ \{\!|Q'\ x|\!\}\ ]\!]$
$\Longrightarrow \{\!|case\text{-}option\ P\ P'\ v|\!\}\ f\ v\ \{\!|\lambda rv.\ case\ v\ of\ None \Rightarrow Q\ rv \mid Some\ x \Rightarrow Q'\ x\ rv|\!\}$
by (*cases* v) *auto*

23.8 Reasoning directly about states

lemma *in-throwError*:
$((v, s') \in fst\ (throwError\ e\ s)) = (v = Inl\ e \wedge s' = s)$
by (*simp add*: *throwError-def return-def*)

lemma *in-returnOk*:
$((v', s') \in fst\ (returnOk\ v\ s)) = (v' = Inr\ v \wedge s' = s)$
by (*simp add*: *returnOk-def return-def*)

lemma *in-bind*:
$((r,s') \in fst\ ((do\ x \leftarrow f;\ g\ x\ od)\ s)) =$
$(\exists s''\ x.\ (x, s'') \in fst\ (f\ s) \wedge (r, s') \in fst\ (g\ x\ s''))$
apply (*simp add*: *bind-def split-def*)

apply *force*
done

lemma *in-bindE-R*:
$((Inr\ r, s') \in fst\ ((doE\ x \leftarrow f;\ g\ x\ odE)\ s)) =$
$(\exists s''\ x.\ (Inr\ x,\ s'') \in fst\ (f\ s) \wedge (Inr\ r,\ s') \in fst\ (g\ x\ s''))$
apply (*simp add*: *bindE-def lift-def split-def bind-def*)
apply (*clarsimp simp*: *throwError-def return-def lift-def split*: *sum.splits*)
apply *safe*
apply (*case-tac a*)
apply *fastforce*
apply *fastforce*
apply *force*
done

lemma *in-bindE-L*:
$((Inl\ r,\ s') \in fst\ ((doE\ x \leftarrow f;\ g\ x\ odE)\ s)) \Longrightarrow$
$(\exists s''\ x.\ (Inr\ x,\ s'') \in fst\ (f\ s) \wedge (Inl\ r,\ s') \in fst\ (g\ x\ s'')) \vee ((Inl\ r,\ s') \in fst\ (f\ s))$
apply (*simp add*: *bindE-def lift-def bind-def*)
apply *safe*
apply (*simp add*: *return-def throwError-def lift-def split-def split*: *sum.splits if-split-asm*)
apply *force*
done

lemma *in-liftE*:
$((v,\ s') \in fst\ (liftE\ f\ s)) = (\exists v'.\ v = Inr\ v' \wedge (v',\ s') \in fst\ (f\ s))$
by (*force simp add*: *liftE-def bind-def return-def split-def*)

lemma *in-whenE*: $((v,\ s') \in fst\ (whenE\ P\ f\ s)) = ((P \longrightarrow (v,\ s') \in fst\ (f\ s)) \wedge$
$(\neg P \longrightarrow v = Inr\ () \wedge s' = s))$
by (*simp add*: *whenE-def in-returnOk*)

lemma *inl-whenE*:
$((Inl\ x,\ s') \in fst\ (whenE\ P\ f\ s)) = (P \wedge (Inl\ x,\ s') \in fst\ (f\ s))$
by (*auto simp add*: *in-whenE*)

lemma *inr-in-unlessE-throwError*[*termination-simp*]:
$(Inr\ (),\ s') \in fst\ (unlessE\ P\ (throwError\ E)\ s) = (P \wedge s'=s)$
by (*simp add*: *unlessE-def returnOk-def throwError-def return-def*)

lemma *in-fail*:
$r \in fst\ (fail\ s) = False$
by (*simp add*: *fail-def*)

lemma *in-return*:
$(r,\ s') \in fst\ (return\ v\ s) = (r = v \wedge s' = s)$
by (*simp add*: *return-def*)

lemma *in-assert*:
  $(r,\ s') \in \mathit{fst}\ (\mathit{assert}\ P\ s) = (P \wedge s' = s)$
  by (*simp add*: *assert-def return-def fail-def*)

lemma *in-assertE*:
  $(r,\ s') \in \mathit{fst}\ (\mathit{assertE}\ P\ s) = (P \wedge r = \mathit{Inr}\ () \wedge s' = s)$
  by (*simp add*: *assertE-def returnOk-def return-def fail-def*)

lemma *in-assert-opt*:
  $(r,\ s') \in \mathit{fst}\ (\mathit{assert\text{-}opt}\ v\ s) = (v = \mathit{Some}\ r \wedge s' = s)$
  by (*auto simp*: *assert-opt-def in-fail in-return split*: *option.splits*)

lemma *in-get*:
  $(r,\ s') \in \mathit{fst}\ (\mathit{get}\ s) = (r = s \wedge s' = s)$
  by (*simp add*: *get-def*)

lemma *in-gets*:
  $(r,\ s') \in \mathit{fst}\ (\mathit{gets}\ f\ s) = (r = f\ s \wedge s' = s)$
  by (*simp add*: *simpler-gets-def*)

lemma *in-put*:
  $(r,\ s') \in \mathit{fst}\ (\mathit{put}\ x\ s) = (s' = x \wedge r = ())$
  by (*simp add*: *put-def*)

lemma *in-when*:
  $(v,\ s') \in \mathit{fst}\ (\mathit{when}\ P\ f\ s) = ((P \longrightarrow (v,\ s') \in \mathit{fst}\ (f\ s)) \wedge (\neg P \longrightarrow v = () \wedge s' = s))$
  by (*simp add*: *when-def in-return*)

lemma *in-modify*:
  $(v,\ s') \in \mathit{fst}\ (\mathit{modify}\ f\ s) = (s'{=}f\ s \wedge v = ())$
  by (*simp add*: *modify-def bind-def get-def put-def*)

lemma *gets-the-in-monad*:
  $((v,\ s') \in \mathit{fst}\ (\mathit{gets\text{-}the}\ f\ s)) = (s' = s \wedge f\ s = \mathit{Some}\ v)$
  by (*auto simp*: *gets-the-def in-bind in-gets in-assert-opt split*: *option.split*)

lemma *in-alternative*:
  $(r,s') \in \mathit{fst}\ ((f \sqcap g)\ s) = ((r,s') \in \mathit{fst}\ (f\ s) \vee (r,s') \in \mathit{fst}\ (g\ s))$
  by (*simp add*: *alternative-def*)

lemmas *in-monad* = *inl-whenE in-whenE in-liftE in-bind in-bindE-L*
    in-bindE-R in-returnOk in-throwError in-fail
    in-assertE in-assert in-return in-assert-opt
    in-get in-gets in-put in-when unlessE-whenE
    unless-when in-modify gets-the-in-monad
    in-alternative

23.9 Non-Failure

lemma *no-failD*:
$\llbracket$ *no-fail P m*; *P s* $\rrbracket \Longrightarrow \neg(snd\ (m\ s))$
by (*simp add*: *no-fail-def*)

lemma *non-fail-modify* [*wp*,*simp*]:
no-fail $\top$ (*modify f*)
by (*simp add*: *no-fail-def modify-def get-def put-def bind-def*)

lemma *non-fail-gets-simp*[*simp*]:
no-fail P (*gets f*)
unfolding *no-fail-def gets-def get-def return-def bind-def*
by *simp*

lemma *non-fail-gets*:
no-fail $\top$ (*gets f*)
by *simp*

lemma *non-fail-select* [*simp*]:
no-fail $\top$ (*select S*)
by (*simp add*: *no-fail-def select-def*)

lemma *no-fail-pre*:
$\llbracket$ *no-fail P f*; $\bigwedge s.\ Q\ s \Longrightarrow P\ s \rrbracket \Longrightarrow$ *no-fail Q f*
by (*simp add*: *no-fail-def*)

lemma *no-fail-alt* [*wp*]:
$\llbracket$ *no-fail P f*; *no-fail Q g* $\rrbracket \Longrightarrow$ *no-fail* (*P and Q*) (*f OR g*)
by (*simp add*: *no-fail-def alternative-def*)

lemma *no-fail-return* [*simp*, *wp*]:
no-fail $\top$ (*return x*)
by (*simp add*: *return-def no-fail-def*)

lemma *no-fail-get* [*simp*, *wp*]:
no-fail $\top$ *get*
by (*simp add*: *get-def no-fail-def*)

lemma *no-fail-put* [*simp*, *wp*]:
no-fail $\top$ (*put s*)
by (*simp add*: *put-def no-fail-def*)

lemma *no-fail-when* [*wp*]:
($P \Longrightarrow$ *no-fail Q f*) $\Longrightarrow$ *no-fail* (*if P then Q else* $\top$) (*when P f*)
by (*simp add*: *when-def*)

lemma *no-fail-unless* [*wp*]:
($\neg P \Longrightarrow$ *no-fail Q f*) $\Longrightarrow$ *no-fail* (*if P then* $\top$ *else Q*) (*unless P f*)
by (*simp add*: *unless-def when-def*)

lemma *no-fail-fail* [*simp*, *wp*]:
  no-fail ⊥ *fail*
  by (*simp add*: *fail-def no-fail-def*)

lemmas [*wp*] = *non-fail-gets*

lemma *no-fail-assert* [*simp*, *wp*]:
  no-fail (λ-. *P*) (*assert P*)
  by (*simp add*: *assert-def*)

lemma *no-fail-assert-opt* [*simp*, *wp*]:
  no-fail (λ-. *P* ≠ *None*) (*assert-opt P*)
  by (*simp add*: *assert-opt-def split*: *option.splits*)

lemma *no-fail-case-option* [*wp*]:
  assumes *f*: *no-fail P f*
  assumes *g*: ⋀*x*. *no-fail* (*Q x*) (*g x*)
  shows *no-fail* (*if x* = *None then P else Q* (*the x*)) (*case-option f g x*)
  by (*clarsimp simp add*: *f g*)

lemma *no-fail-if* [*wp*]:
  ⟦ *P* ⟹ *no-fail Q f*; ¬*P* ⟹ *no-fail R g* ⟧ ⟹
  no-fail (*if P then Q else R*) (*if P then f else g*)
  by *simp*

lemma *no-fail-apply* [*wp*]:
  no-fail P (*f* (*g x*)) ⟹ *no-fail P* (*f* $ *g x*)
  by *simp*

lemma *no-fail-undefined* [*simp*, *wp*]:
  no-fail ⊥ *undefined*
  by (*simp add*: *no-fail-def*)

lemma *no-fail-returnOK* [*simp*, *wp*]:
  no-fail ⊤ (*returnOk x*)
  by (*simp add*: *returnOk-def*)

lemma *no-fail-bind* [*wp*]:
  assumes *f*: *no-fail P f*
  assumes *g*: ⋀*rv*. *no-fail* (*R rv*) (*g rv*)
  assumes *v*: ⦃*Q*⦄ *f* ⦃*R*⦄
  shows *no-fail* (*P and Q*) (*f* >>= (λ*rv*. *g rv*))
  apply (*clarsimp simp*: *no-fail-def bind-def*)
  apply (*rule conjI*)
   prefer *2*
   apply (*erule no-failD* [*OF f*])
  apply *clarsimp*
  apply (*drule* (*1*) *use-valid* [*OF* - *v*])

apply (*drule no-failD* [*OF g*])
apply *simp*
done

Empty results implies non-failure

lemma *empty-fail-modify* [*simp*, *wp*]:
empty-fail (*modify f*)
by (*simp add*: *empty-fail-def simpler-modify-def*)

lemma *empty-fail-gets* [*simp*, *wp*]:
empty-fail (*gets f*)
by (*simp add*: *empty-fail-def simpler-gets-def*)

lemma *empty-failD*:
$\llbracket$ *empty-fail m*; *fst* (*m s*) = {} $\rrbracket \Longrightarrow$ *snd* (*m s*)
by (*simp add*: *empty-fail-def*)

lemma *empty-fail-select-f* [*simp*]:
assumes *ef*: *fst S* = {} $\Longrightarrow$ *snd S*
shows *empty-fail* (*select-f S*)
by (*fastforce simp add*: *empty-fail-def select-f-def intro*: *ef*)

lemma *empty-fail-bind* [*simp*]:
$\llbracket$ *empty-fail a*; $\bigwedge$*x*. *empty-fail* (*b x*) $\rrbracket \Longrightarrow$ *empty-fail* (*a* >>= *b*)
apply (*simp add*: *bind-def empty-fail-def split-def*)
apply *clarsimp*
apply (*case-tac fst* (*a s*) = {})
 apply *blast*
apply (*clarsimp simp*: *ex-in-conv* [*symmetric*])
done

lemma *empty-fail-return* [*simp*, *wp*]:
empty-fail (*return x*)
by (*simp add*: *empty-fail-def return-def*)

lemma *empty-fail-mapM* [*simp*]:
assumes *m*: $\bigwedge$*x*. *empty-fail* (*m x*)
shows *empty-fail* (*mapM m xs*)
proof (*induct xs*)
case *Nil*
thus *?case* **by** (*simp add*: *mapM-def sequence-def*)
next
case *Cons*
have *P*: $\bigwedge$*m x xs*. *mapM m* (*x* # *xs*) = (*do y* ← *m x*; *ys* ← (*mapM m xs*); *return* (*y* # *ys*) *od*)
by (*simp add*: *mapM-def sequence-def Let-def*)
from *Cons*
show *?case* **by** (*simp add*: *P m*)
qed

lemma *empty-fail* [*simp*]:
empty-fail fail
by (*simp add*: *fail-def empty-fail-def*)

lemma *empty-fail-assert-opt* [*simp*]:
empty-fail (*assert-opt x*)
by (*simp add*: *assert-opt-def split*: *option.splits*)

lemma *empty-fail-mk-ef*:
empty-fail (*mk-ef o m*)
by (*simp add*: *empty-fail-def mk-ef-def*)

lemma *empty-fail-gets-map*[*simp*]:
empty-fail (*gets-map f p*)
unfolding *gets-map-def* **by** *simp*

23.10 Failure

lemma *fail-wp*: $\{\!|\lambda x.\ True|\!\}$ *fail* $\{\!|Q|\!\}$
by (*simp add*: *valid-def fail-def*)

lemma *failE-wp*: $\{\!|\lambda x.\ True|\!\}$ *fail* $\{\!|Q|\!\},\{\!|E|\!\}$
by (*simp add*: *validE-def fail-wp*)

lemma *fail-update* [*iff*]:
fail (*f s*) = *fail s*
by (*simp add*: *fail-def*)

We can prove postconditions using hoare triples

lemma *post-by-hoare*: $[\![\ \{\!|P|\!\}\ f\ \{\!|Q|\!\};\ P\ s;\ (r,\ s') \in fst\ (f\ s)\ ]\!] \Longrightarrow Q\ r\ s'$
apply (*simp add*: *valid-def*)
apply *blast*
done

Weakest Precondition Rules

lemma *hoare-vcg-prop*:
$\{\!|\lambda s.\ P|\!\}\ f\ \{\!|\lambda rv\ s.\ P|\!\}$
by (*simp add*: *valid-def*)

lemma *return-wp*:
$\{\!|P\ x|\!\}$ *return x* $\{\!|P|\!\}$
by(*simp add*:*valid-def return-def*)

lemma *get-wp*:
$\{\!|\lambda s.\ P\ s\ s|\!\}$ *get* $\{\!|P|\!\}$
by(*simp add*:*valid-def split-def get-def*)

lemma *gets-wp*:

$\{\!|\lambda s.\ P\ (f\ s)\ s|\!\}$ *gets f* $\{\!|P|\!\}$
by(*simp add*:*valid-def split-def gets-def return-def get-def bind-def*)

lemma *modify-wp*:
$\{\!|\lambda s.\ P\ ()\ (f\ s)|\!\}$ *modify f* $\{\!|P|\!\}$
by(*simp add*:*valid-def split-def modify-def get-def put-def bind-def*)

lemma *put-wp*:
$\{\!|\lambda s.\ P\ ()\ x|\!\}$ *put x* $\{\!|P|\!\}$
by(*simp add*:*valid-def put-def*)

lemma *returnOk-wp*:
$\{\!|P\ x|\!\}$ *returnOk x* $\{\!|P|\!\},\{\!|E|\!\}$
by(*simp add*:*validE-def2 returnOk-def return-def*)

lemma *throwError-wp*:
$\{\!|E\ e|\!\}$ *throwError e* $\{\!|P|\!\},\{\!|E|\!\}$
by(*simp add*:*validE-def2 throwError-def return-def*)

lemma *returnOKE-R-wp* : $\{\!|P\ x|\!\}$ *returnOk x* $\{\!|P|\!\}, -$
by (*simp add*: *validE-R-def validE-def valid-def returnOk-def return-def*)

lemma *liftE-wp*:
$\{\!|P|\!\}\ f\ \{\!|Q|\!\} \Longrightarrow \{\!|P|\!\}$ *liftE f* $\{\!|Q|\!\},\{\!|E|\!\}$
by(*clarsimp simp*:*valid-def validE-def2 liftE-def split-def Let-def bind-def return-def*)

lemma *catch-wp*:
$[\![\ \bigwedge x.\ \{\!|E\ x|\!\}$ *handler x* $\{\!|Q|\!\};\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\}\ ]\!] \Longrightarrow$
$\{\!|P|\!\}$ *catch f handler* $\{\!|Q|\!\}$
apply (*unfold catch-def valid-def validE-def return-def*)
apply (*fastforce simp*: *bind-def split*: *sum.splits*)
done

lemma *handleE′-wp*:
$[\![\ \bigwedge x.\ \{\!|F\ x|\!\}$ *handler x* $\{\!|Q|\!\},\{\!|E|\!\};\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|F|\!\}\ ]\!] \Longrightarrow$
$\{\!|P|\!\}$ *f <handle2> handler* $\{\!|Q|\!\},\{\!|E|\!\}$
apply (*unfold handleE′-def valid-def validE-def return-def*)
apply (*fastforce simp*: *bind-def split*: *sum.splits*)
done

lemma *handleE-wp*:
assumes *x*: $\bigwedge x.\ \{\!|F\ x|\!\}$ *handler x* $\{\!|Q|\!\},\{\!|E|\!\}$
assumes *y*: $\{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|F|\!\}$
shows $\{\!|P|\!\}$ *f <handle> handler* $\{\!|Q|\!\},\{\!|E|\!\}$
by (*simp add*: *handleE-def handleE′-wp* [*OF x y*])

lemma *hoare-vcg-if-split*:
$[\![\ P \Longrightarrow \{\!|Q|\!\}\ f\ \{\!|S|\!\};\ \neg P \Longrightarrow \{\!|R|\!\}\ g\ \{\!|S|\!\}\ ]\!] \Longrightarrow$
$\{\!|\lambda s.\ (P \longrightarrow Q\ s) \wedge (\neg P \longrightarrow R\ s)|\!\}$ *if P then f else g* $\{\!|S|\!\}$

by *simp*

lemma *hoare-vcg-if-splitE*:
$\llbracket\ P \Longrightarrow \{\!|Q|\!\}\ f\ \{\!|S|\!\},\{\!|E|\!\};\ \neg P \Longrightarrow \{\!|R|\!\}\ g\ \{\!|S|\!\},\{\!|E|\!\}\ \rrbracket \Longrightarrow$
$\{\!|\lambda s.\ (P \longrightarrow Q\ s) \wedge (\neg P \longrightarrow R\ s)|\!\}$ *if P then f else g* $\{\!|S|\!\},\{\!|E|\!\}$
by *simp*

lemma *hoare-liftM-subst*: $\{\!|P|\!\}$ *liftM f m* $\{\!|Q|\!\} = \{\!|P|\!\}\ m\ \{\!|Q \circ f|\!\}$
apply (*simp add*: *liftM-def bind-def return-def split-def*)
apply (*simp add*: *valid-def Ball-def*)
apply (*rule-tac f=All* **in** *arg-cong*)
apply (*rule ext*)
apply *fastforce*
done

lemma *liftE-validE*[*simp*]: $\{\!|P|\!\}$ *liftE f* $\{\!|Q|\!\},\{\!|E|\!\} = \{\!|P|\!\}\ f\ \{\!|Q|\!\}$
apply (*simp add*: *liftE-liftM validE-def hoare-liftM-subst o-def*)
done

lemma *liftM-wp*: $\{\!|P|\!\}\ m\ \{\!|Q \circ f|\!\} \Longrightarrow \{\!|P|\!\}$ *liftM f m* $\{\!|Q|\!\}$
by (*simp add*: *hoare-liftM-subst*)

lemma *hoare-liftME-subst*: $\{\!|P|\!\}$ *liftME f m* $\{\!|Q|\!\},\{\!|E|\!\} = \{\!|P|\!\}\ m\ \{\!|Q \circ f|\!\},\{\!|E|\!\}$
apply (*simp add*: *validE-def liftME-liftM hoare-liftM-subst o-def*)
apply (*rule-tac f=valid P m* **in** *arg-cong*)
apply (*rule ext*)+
apply (*case-tac x*, *simp-all*)
done

lemma *liftME-wp*: $\{\!|P|\!\}\ m\ \{\!|Q \circ f|\!\},\{\!|E|\!\} \Longrightarrow \{\!|P|\!\}$ *liftME f m* $\{\!|Q|\!\},\{\!|E|\!\}$
by (*simp add*: *hoare-liftME-subst*)

lemma *o-const-simp*[*simp*]: $(\lambda x.\ C) \circ f = (\lambda x.\ C)$
by (*simp add*: *o-def*)

lemma *hoare-vcg-split-case-option*:
$\llbracket\ \bigwedge x.\ x = None \Longrightarrow \{\!|P\ x|\!\}\ f\ x\ \{\!|R\ x|\!\};$
$\quad \bigwedge x\ y.\ x = Some\ y \Longrightarrow \{\!|Q\ x\ y|\!\}\ g\ x\ y\ \{\!|R\ x|\!\}\ \rrbracket \Longrightarrow$
$\{\!|\lambda s.\ (x = None \longrightarrow P\ x\ s) \wedge$
$\qquad (\forall y.\ x = Some\ y \longrightarrow Q\ x\ y\ s)|\!\}$
case x of None $\Rightarrow$ *f x*
$\qquad$| *Some y* $\Rightarrow$ *g x y*
$\{\!|R\ x|\!\}$
apply(*simp add*:*valid-def split-def*)
apply(*case-tac x*, *simp-all*)
done

lemma *hoare-vcg-split-case-optionE*:

assumes *none-case*: $\bigwedge x.\ x = None \Longrightarrow \{\!|P\ x|\!\}\ f\ x\ \{\!|R\ x|\!\},\{\!|E\ x|\!\}$
assumes *some-case*: $\bigwedge x\ y.\ x = Some\ y \Longrightarrow \{\!|Q\ x\ y|\!\}\ g\ x\ y\ \{\!|R\ x|\!\},\{\!|E\ x|\!\}$
shows $\{\!|\lambda s.\ (x = None \longrightarrow P\ x\ s) \wedge$
$(\forall y.\ x = Some\ y \longrightarrow Q\ x\ y\ s)|\!\}$
case x of None $\Rightarrow$ *f x*
$|$ *Some y* $\Rightarrow$ *g x y*
$\{\!|R\ x|\!\},\{\!|E\ x|\!\}$
apply(*case-tac x*, *simp-all*)
apply(*rule none-case*, *simp*)
apply(*rule some-case*, *simp*)
done

lemma *hoare-vcg-split-case-sum*:
$[\![\ \bigwedge x\ a.\ x = Inl\ a \Longrightarrow \{\!|P\ x\ a|\!\}\ f\ x\ a\ \{\!|R\ x|\!\};$
$\bigwedge x\ b.\ x = Inr\ b \Longrightarrow \{\!|Q\ x\ b|\!\}\ g\ x\ b\ \{\!|R\ x|\!\}\ ]\!] \Longrightarrow$
$\{\!|\lambda s.\ (\forall a.\ x = Inl\ a \longrightarrow P\ x\ a\ s) \wedge$
$(\forall b.\ x = Inr\ b \longrightarrow Q\ x\ b\ s)\ |\!\}$
case x of Inl a $\Rightarrow$ *f x a*
$|$ *Inr b* $\Rightarrow$ *g x b*
$\{\!|R\ x|\!\}$
apply(*simp add*:*valid-def split-def*)
apply(*case-tac x*, *simp-all*)
done

lemma *hoare-vcg-split-case-sumE*:
assumes *left-case*: $\bigwedge x\ a.\ x = Inl\ a \Longrightarrow \{\!|P\ x\ a|\!\}\ f\ x\ a\ \{\!|R\ x|\!\}$
assumes *right-case*: $\bigwedge x\ b.\ x = Inr\ b \Longrightarrow \{\!|Q\ x\ b|\!\}\ g\ x\ b\ \{\!|R\ x|\!\}$
shows $\{\!|\lambda s.\ (\forall a.\ x = Inl\ a \longrightarrow P\ x\ a\ s) \wedge$
$(\forall b.\ x = Inr\ b \longrightarrow Q\ x\ b\ s)\ |\!\}$
case x of Inl a $\Rightarrow$ *f x a*
$|$ *Inr b* $\Rightarrow$ *g x b*
$\{\!|R\ x|\!\}$
apply(*case-tac x*, *simp-all*)
apply(*rule left-case*, *simp*)
apply(*rule right-case*, *simp*)
done

lemma *hoare-vcg-precond-imp*:
$[\![\ \{\!|Q|\!\}\ f\ \{\!|R|\!\};\ \bigwedge s.\ P\ s \Longrightarrow Q\ s\ ]\!] \Longrightarrow \{\!|P|\!\}\ f\ \{\!|R|\!\}$
by (*fastforce simp add*:*valid-def*)

lemma *hoare-vcg-precond-impE*:
$[\![\ \{\!|Q|\!\}\ f\ \{\!|R|\!\},\{\!|E|\!\};\ \bigwedge s.\ P\ s \Longrightarrow Q\ s\ ]\!] \Longrightarrow \{\!|P|\!\}\ f\ \{\!|R|\!\},\{\!|E|\!\}$
by (*fastforce simp add*:*validE-def2*)

lemma *hoare-seq-ext*:
assumes *g-valid*: $\bigwedge x.\ \{\!|B\ x|\!\}\ g\ x\ \{\!|C|\!\}$
assumes *f-valid*: $\{\!|A|\!\}\ f\ \{\!|B|\!\}$
shows $\{\!|A|\!\}\ do\ x \leftarrow f;\ g\ x\ od\ \{\!|C|\!\}$

apply(*insert f-valid g-valid*)
apply(*blast intro*: *seq-ext'*)
done

lemma *hoare-vcg-seqE*:
assumes *g-valid*: $\bigwedge x.\ \{\!|B\ x|\!\}\ g\ x\ \{\!|C|\!\},\{\!|E|\!\}$
assumes *f-valid*: $\{\!|A|\!\}\ f\ \{\!|B|\!\},\{\!|E|\!\}$
shows $\{\!|A|\!\}\ doE\ x \leftarrow f;\ g\ x\ odE\ \{\!|C|\!\},\{\!|E|\!\}$
apply(*insert f-valid g-valid*)
apply(*blast intro*: *seqE'*)
done

lemma *hoare-seq-ext-nobind*:
$[\![\ \{\!|B|\!\}\ g\ \{\!|C|\!\};$
$\{\!|A|\!\}\ f\ \{\!|\lambda r\ s.\ B\ s|\!\}\ ]\!] \Longrightarrow$
$\{\!|A|\!\}\ do\ f;\ g\ od\ \{\!|C|\!\}$
apply (*clarsimp simp*: *valid-def bind-def Let-def split-def*)
apply *fastforce*
done

lemma *hoare-seq-ext-nobindE*:
$[\![\ \{\!|B|\!\}\ g\ \{\!|C|\!\},\{\!|E|\!\};$
$\{\!|A|\!\}\ f\ \{\!|\lambda r\ s.\ B\ s|\!\},\{\!|E|\!\}\ ]\!] \Longrightarrow$
$\{\!|A|\!\}\ doE\ f;\ g\ odE\ \{\!|C|\!\},\{\!|E|\!\}$
apply (*clarsimp simp*:*validE-def*)
apply (*simp add*:*bindE-def Let-def split-def bind-def lift-def*)
apply (*fastforce simp add*: *valid-def throwError-def return-def lift-def*
split: *sum.splits*)
done

lemma *hoare-chain*:
$[\![\ \{\!|P|\!\}\ f\ \{\!|Q|\!\};$
$\bigwedge s.\ R\ s \Longrightarrow P\ s;$
$\bigwedge r\ s.\ Q\ r\ s \Longrightarrow S\ r\ s\ ]\!] \Longrightarrow$
$\{\!|R|\!\}\ f\ \{\!|S|\!\}$
by(*fastforce simp add*:*valid-def split-def*)

lemma *validE-weaken*:
$[\![\ \{\!|P'|\!\}\ A\ \{\!|Q'|\!\},\{\!|E'|\!\};\ \bigwedge s.\ P\ s \Longrightarrow P'\ s;\ \bigwedge r\ s.\ Q'\ r\ s \Longrightarrow Q\ r\ s;\ \bigwedge r\ s.\ E'\ r\ s$
$\Longrightarrow E\ r\ s\ ]\!] \Longrightarrow \{\!|P|\!\}\ A\ \{\!|Q|\!\},\{\!|E|\!\}$
by (*fastforce simp*: *validE-def2 split*: *sum.splits*)

lemmas *hoare-chainE* = *validE-weaken*

lemma *hoare-vcg-handle-elseE*:
$[\![\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\};$
$\bigwedge e.\ \{\!|E\ e|\!\}\ g\ e\ \{\!|R|\!\},\{\!|F|\!\};$
$\bigwedge x.\ \{\!|Q\ x|\!\}\ h\ x\ \{\!|R|\!\},\{\!|F|\!\}\ ]\!] \Longrightarrow$
$\{\!|P|\!\}$ *f* <*handle*> *g* <*else*> *h* $\{\!|R|\!\},\{\!|F|\!\}$

apply (*simp add*: *handle-elseE-def validE-def*)
apply (*rule seq-ext*)
 apply *assumption*
apply (*case-tac x*, *simp-all*)
done

lemma *alternative-valid*:
assumes *x*: $\{\!|P|\!\}$ *f* $\{\!|Q|\!\}$
assumes *y*: $\{\!|P|\!\}$ f' $\{\!|Q|\!\}$
shows $\{\!|P|\!\}$ *f OR* f' $\{\!|Q|\!\}$
apply (*simp add*: *valid-def alternative-def*)
apply *safe*
 apply (*simp add*: *post-by-hoare* [*OF x*])
apply (*simp add*: *post-by-hoare* [*OF y*])
done

lemma *alternative-wp*:
assumes *x*: $\{\!|P|\!\}$ *f* $\{\!|Q|\!\}$
assumes *y*: $\{\!|P'|\!\}$ f' $\{\!|Q|\!\}$
shows $\{\!|P \textit{ and } P'|\!\}$ *f OR* f' $\{\!|Q|\!\}$
apply (*rule alternative-valid*)
 apply (*rule hoare-pre-imp* [*OF - x*], *simp*)
apply (*rule hoare-pre-imp* [*OF - y*], *simp*)
done

lemma *alternativeE-wp*:
assumes *x*: $\{\!|P|\!\}$ *f* $\{\!|Q|\!\},\{\!|E|\!\}$ **and** *y*: $\{\!|P'|\!\}$ f' $\{\!|Q|\!\},\{\!|E|\!\}$
shows $\{\!|P \textit{ and } P'|\!\}$ *f OR* f' $\{\!|Q|\!\},\{\!|E|\!\}$
apply (*unfold validE-def*)
apply (*wp add*: *x y alternative-wp* | *simp* | *fold validE-def*)+
done

lemma *alternativeE-R-wp*:
$[\![\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},-;\ \{\!|P'|\!\}\ f'\ \{\!|Q|\!\},-\ ]\!] \Longrightarrow \{\!|P \textit{ and } P'|\!\}\ f\ \textit{OR}\ f'\ \{\!|Q|\!\},-$
apply (*simp add*: *validE-R-def*)
apply (*rule alternativeE-wp*)
 apply *assumption*+
done

lemma *alternative-R-wp*:
$[\![\ \{\!|P|\!\}\ f\ -,\{\!|Q|\!\};\ \{\!|P'|\!\}\ g\ -,\{\!|Q|\!\}\ ]\!] \Longrightarrow \{\!|P \textit{ and } P'|\!\}\ f \sqcap g\ -,\ \{\!|Q|\!\}$
by (*fastforce simp*: *alternative-def validE-E-def validE-def valid-def*)

lemma *select-wp*: $\{\!|\lambda s.\ \forall x \in S.\ Q\ x\ s|\!\}$ *select S* $\{\!|Q|\!\}$
by (*simp add*: *select-def valid-def*)

lemma *select-f-wp*:
$\{\!|\lambda s.\ \forall x{\in}\textit{fst}\ S.\ Q\ x\ s|\!\}$ *select-f S* $\{\!|Q|\!\}$
by (*simp add*: *select-f-def valid-def*)

lemma *state-select-wp* [*wp*]: $\{\!|\ \lambda s.\ \forall t.\ (s,\ t) \in f \longrightarrow P\ ()\ t\ |\!\}$ *state-select f* $\{\!|\ P\ |\!\}$
  apply (*clarsimp simp*: *state-select-def*)
  apply (*clarsimp simp*: *valid-def*)
  done

lemma *condition-wp* [*wp*]:
  $[\![\ \{\!|\ Q\ |\!\}\ A\ \{\!|\ P\ |\!\};\ \{\!|\ R\ |\!\}\ B\ \{\!|\ P\ |\!\}\ ]\!] \Longrightarrow \{\!|\ \lambda s.\ \mathit{if}\ C\ s\ \mathit{then}\ Q\ s\ \mathit{else}\ R\ s\ |\!\}$ *condition* $C\ A\ B\ \{\!|\ P\ |\!\}$
  apply (*clarsimp simp*: *condition-def*)
  apply (*clarsimp simp*: *valid-def pred-conj-def pred-neg-def split-def*)
  done

lemma *conditionE-wp* [*wp*]:
  $[\![\ \{\!|\ P\ |\!\}\ A\ \{\!|\ Q\ |\!\},\{\!|\ R\ |\!\};\ \{\!|\ P'\ |\!\}\ B\ \{\!|\ Q\ |\!\},\{\!|\ R\ |\!\}\ ]\!] \Longrightarrow\ \{\!|\ \lambda s.\ \mathit{if}\ C\ s\ \mathit{then}\ P\ s\ \mathit{else}\ P'\ s\ |\!\}$ *condition* $C\ A\ B\ \{\!|Q|\!\},\{\!|R|\!\}$
  apply (*clarsimp simp*: *condition-def*)
  apply (*clarsimp simp*: *validE-def valid-def*)
  done

lemma *state-assert-wp* [*wp*]: $\{\!|\ \lambda s.\ f\ s \longrightarrow P\ ()\ s\ |\!\}$ *state-assert f* $\{\!|\ P\ |\!\}$
  apply (*clarsimp simp*: *state-assert-def get-def*
    assert-def bind-def valid-def return-def fail-def)
  done

The weakest precondition handler which works on conjunction

lemma *hoare-vcg-conj-lift*:
  assumes *x*: $\{\!|P|\!\}\ f\ \{\!|Q|\!\}$
  assumes *y*: $\{\!|P'|\!\}\ f\ \{\!|Q'|\!\}$
  shows $\{\!|\lambda s.\ P\ s \wedge P'\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s|\!\}$
  apply (*subst bipred-conj-def*[*symmetric*], *rule hoare-post-conj*)
   apply (*rule hoare-pre-imp* [*OF - x*], *simp*)
  apply (*rule hoare-pre-imp* [*OF - y*], *simp*)
  done

lemma *hoare-vcg-conj-liftE1*:
  $[\![\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},-;\ \{\!|P'|\!\}\ f\ \{\!|Q'|\!\},\{\!|E|\!\}\ ]\!] \Longrightarrow$
  $\{\!|P\ \mathit{and}\ P'|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ r\ s \wedge Q'\ r\ s|\!\},\{\!|E|\!\}$
  unfolding *valid-def validE-R-def validE-def*
  apply (*clarsimp simp*: *split-def split*: *sum.splits*)
  apply (*erule allE*, *erule* (*1*) *impE*)
  apply (*erule allE*, *erule* (*1*) *impE*)
  apply (*drule* (*1*) *bspec*)
  apply (*drule* (*1*) *bspec*)
  apply *clarsimp*
  done

lemma *hoare-vcg-disj-lift*:
  assumes *x*: $\{\!|P|\!\}\ \ f\ \{\!|Q|\!\}$

assumes y: $\{\!|P'|\!\}$ f $\{\!|Q'|\!\}$
shows $\{\!|\lambda s.\ P\ s \vee P'\ s|\!\}$ f $\{\!|\lambda rv\ s.\ Q\ rv\ s \vee Q'\ rv\ s|\!\}$
apply (*simp add*: *valid-def*)
apply *safe*
apply (*erule*(*1*) *post-by-hoare* [*OF x*])
apply (*erule notE*)
apply (*erule*(*1*) *post-by-hoare* [*OF y*])
done

lemma *hoare-vcg-const-Ball-lift*:
$[\![\bigwedge x.\ x \in S \Longrightarrow \{\!|P\ x|\!\}\ f\ \{\!|Q\ x|\!\}]\!] \Longrightarrow \{\!|\lambda s.\ \forall x{\in}S.\ P\ x\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ \forall x{\in}S.\ Q\ x\ rv\ s|\!\}$
by (*fastforce simp*: *valid-def*)

lemma *hoare-vcg-const-Ball-lift-R*:
$[\![\bigwedge x.\ x \in S \Longrightarrow \{\!|P\ x|\!\}\ f\ \{\!|Q\ x|\!\},-]\!] \Longrightarrow$
$\{\!|\lambda s.\ \forall x \in S.\ P\ x\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ \forall x \in S.\ Q\ x\ rv\ s|\!\},-$
apply (*simp add*: *validE-R-def validE-def*)
apply (*rule hoare-strengthen-post*)
apply (*erule hoare-vcg-const-Ball-lift*)
apply (*simp split*: *sum.splits*)
done

lemma *hoare-vcg-all-lift*:
$[\![\bigwedge x.\ \{\!|P\ x|\!\}\ f\ \{\!|Q\ x|\!\}]\!] \Longrightarrow \{\!|\lambda s.\ \forall x.\ P\ x\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ \forall x.\ Q\ x\ rv\ s|\!\}$
by (*fastforce simp*: *valid-def*)

lemma *hoare-vcg-all-lift-R*:
$(\bigwedge x.\ \{\!|P\ x|\!\}\ f\ \{\!|Q\ x|\!\}, -) \Longrightarrow \{\!|\lambda s.\ \forall x.\ P\ x\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ \forall x.\ Q\ x\ rv\ s|\!\}, -$
by (*rule hoare-vcg-const-Ball-lift-R*[**where** S=*UNIV*, *simplified*])

lemma *hoare-vcg-imp-lift*:
$[\![\{\!|P'|\!\}\ f\ \{\!|\lambda rv\ s.\ \neg\ P\ rv\ s|\!\};\ \{\!|Q'|\!\}\ f\ \{\!|Q|\!\}]\!] \Longrightarrow \{\!|\lambda s.\ P'\ s \vee Q'\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ P\ rv\ s \longrightarrow Q\ rv\ s|\!\}$
apply (*simp only*: *imp-conv-disj*)
apply (*erule*(*1*) *hoare-vcg-disj-lift*)
done

lemma *hoare-vcg-imp-lift'*:
$[\![\{\!|P'|\!\}\ f\ \{\!|\lambda rv\ s.\ \neg\ P\ rv\ s|\!\};\ \{\!|Q'|\!\}\ f\ \{\!|Q|\!\}]\!] \Longrightarrow \{\!|\lambda s.\ \neg\ P'\ s \longrightarrow Q'\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ P\ rv\ s \longrightarrow Q\ rv\ s|\!\}$
apply (*simp only*: *imp-conv-disj*)
apply *simp*
apply (*erule* (*1*) *hoare-vcg-imp-lift*)
done

lemma *hoare-vcg-imp-conj-lift*[*wp-comb*]:
$\{\!|P|\!\}\ f\ \{\!|\lambda rv\ s.\ Q\ rv\ s \longrightarrow Q'\ rv\ s|\!\} \Longrightarrow \{\!|P'|\!\}\ f\ \{\!|\lambda rv\ s.\ (Q\ rv\ s \longrightarrow Q''\ rv\ s) \wedge$

$Q''' \ rv \ s\}$
$\Longrightarrow \{P \ and \ P'\} \ f \ \{\lambda rv \ s. \ (Q \ rv \ s \longrightarrow Q' \ rv \ s \wedge Q'' \ rv \ s) \wedge Q''' \ rv \ s\}$
by (*auto simp*: *valid-def*)

lemmas *hoare-vcg-imp-conj-lift'*[*wp-unsafe*] = *hoare-vcg-imp-conj-lift*[**where** Q'''=$\top\top$, *simplified*]

lemma *hoare-absorb-imp*:
$\{ P \} \ f \ \{\lambda rv \ s. \ Q \ rv \ s \wedge R \ rv \ s \ \} \Longrightarrow \{ P \} \ f \ \{\lambda rv \ s. \ Q \ rv \ s \longrightarrow R \ rv \ s \}$
by (*erule hoare-post-imp*[*rotated*], *blast*)

lemma *hoare-weaken-imp*:
$[\![\bigwedge rv \ s. \ Q \ rv \ s \Longrightarrow Q' \ rv \ s \ ; \ \{P\} \ f \ \{\lambda rv \ s. \ Q' \ rv \ s \longrightarrow R \ rv \ s\} \ ]\!]$
$\Longrightarrow \{P\} \ f \ \{\lambda rv \ s. \ Q \ rv \ s \longrightarrow R \ rv \ s\}$
by (*clarsimp simp*: *NonDetMonad.valid-def split-def*)

lemma *hoare-vcg-const-imp-lift*:
$[\![P \Longrightarrow \{Q\} \ m \ \{R\} \ ]\!] \Longrightarrow$
$\{\lambda s. \ P \longrightarrow Q \ s\} \ m \ \{\lambda rv \ s. \ P \longrightarrow R \ rv \ s\}$
by (*cases P*, *simp-all add*: *hoare-vcg-prop*)

lemma *hoare-vcg-const-imp-lift-R*:
$(P \Longrightarrow \{Q\} \ m \ \{R\},-) \Longrightarrow \{\lambda s. \ P \longrightarrow Q \ s\} \ m \ \{\lambda rv \ s. \ P \longrightarrow R \ rv \ s\},-$
by (*fastforce simp*: *validE-R-def validE-def valid-def split-def split*: *sum.splits*)

lemma *hoare-weak-lift-imp*:
$\{P'\} \ f \ \{Q\} \Longrightarrow \{\lambda s. \ P \longrightarrow P' \ s\} \ f \ \{\lambda rv \ s. \ P \longrightarrow Q \ rv \ s\}$
by (*auto simp add*: *valid-def split-def*)

lemma *hoare-vcg-weaken-imp*:
$[\![\bigwedge rv \ s. \ Q \ rv \ s \Longrightarrow Q' \ rv \ s \ ; \ \{ P \} \ f \ \{\lambda rv \ s. \ Q' \ rv \ s \longrightarrow R \ rv \ s\} \ ]\!]$
$\Longrightarrow \{ P \} \ f \ \{\lambda rv \ s. \ Q \ rv \ s \longrightarrow R \ rv \ s\}$
by (*clarsimp simp*: *valid-def split-def*)

lemma *hoare-vcg-ex-lift*:
$[\![\bigwedge x. \ \{P \ x\} \ f \ \{Q \ x\} \ ]\!] \Longrightarrow \{\lambda s. \ \exists x. \ P \ x \ s\} \ f \ \{\lambda rv \ s. \ \exists x. \ Q \ x \ rv \ s\}$
by (*clarsimp simp*: *valid-def*, *blast*)

lemma *hoare-vcg-ex-lift-R1*:
$(\bigwedge x. \ \{P \ x\} \ f \ \{Q\}, \ -) \Longrightarrow \{\lambda s. \ \exists x. \ P \ x \ s\} \ f \ \{Q\}, \ -$
by (*fastforce simp*: *valid-def validE-R-def validE-def split*: *sum.splits*)

lemma *hoare-liftP-ext*:
assumes $\bigwedge P \ x. \ m \ \{\lambda s. \ P \ (f \ s \ x)\}$
shows $m \ \{\lambda s. \ P \ (f \ s)\}$
unfolding *valid-def*
apply *clarsimp*
apply (*erule rsubst*[**where** P=P])
apply (*rule ext*)

apply (*drule use-valid*, *rule assms*, *rule refl*)
apply *simp*
done

lemma *hoare-triv*: $\{\!|P|\!\}f\{\!|Q|\!\} \Longrightarrow \{\!|P|\!\}f\{\!|Q|\!\}$.
lemma *hoare-trivE*: $\{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\}$.
lemma *hoare-trivE-R*: $\{\!|P|\!\}\ f\ \{\!|Q|\!\},- \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},-$.
lemma *hoare-trivR-R*: $\{\!|P|\!\}\ f\ -,\{\!|E|\!\} \Longrightarrow \{\!|P|\!\}\ f\ -,\{\!|E|\!\}$.

lemma *hoare-weaken-preE-E*:
$[\![\ \{\!|P'|\!\}\ f\ -,\{\!|Q|\!\};\ \bigwedge s.\ P\ s \Longrightarrow P'\ s\ ]\!] \Longrightarrow \{\!|P|\!\}\ f\ -,\{\!|Q|\!\}$
by (*fastforce simp add*: *validE-E-def validE-def valid-def*)

lemma *hoare-vcg-E-conj*:
$[\![\ \{\!|P|\!\}\ f\ -,\{\!|E|\!\};\ \{\!|P'|\!\}\ f\ \{\!|Q'|\!\},\{\!|E'|\!\}\ ]\!]$
$\Longrightarrow \{\!|\lambda s.\ P\ s \wedge P'\ s|\!\}\ f\ \{\!|Q'|\!\},\ \{\!|\lambda rv\ s.\ E\ rv\ s \wedge E'\ rv\ s|\!\}$
apply (*unfold validE-def validE-E-def*)
apply (*rule hoare-post-imp* [*OF - hoare-vcg-conj-lift*], *simp-all*)
apply (*case-tac r*, *simp-all*)
done

lemma *hoare-vcg-E-elim*:
$[\![\ \{\!|P|\!\}\ f\ -,\{\!|E|\!\};\ \{\!|P'|\!\}\ f\ \{\!|Q|\!\},-\ ]\!]$
$\Longrightarrow \{\!|\lambda s.\ P\ s \wedge P'\ s|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\}$
by (*rule hoare-post-impErr* [*OF hoare-vcg-E-conj*],
(*simp add*: *validE-R-def*)+)

lemma *hoare-vcg-R-conj*:
$[\![\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},-;\ \{\!|P'|\!\}\ f\ \{\!|Q'|\!\},-\ ]\!]$
$\Longrightarrow \{\!|\lambda s.\ P\ s \wedge P'\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s|\!\},-$
apply (*unfold validE-R-def validE-def*)
apply (*rule hoare-post-imp* [*OF - hoare-vcg-conj-lift*], *simp-all*)
apply (*case-tac r*, *simp-all*)
done

lemma *valid-validE*:
$\{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q|\!\},\{\!|\lambda rv.\ Q|\!\}$
apply (*simp add*: *validE-def*)
done

lemma *valid-validE2*:
$[\![\ \{\!|P|\!\}\ f\ \{\!|\lambda\text{-}.\ Q'|\!\};\ \bigwedge s.\ Q'\ s \Longrightarrow Q\ s;\ \bigwedge s.\ Q'\ s \Longrightarrow E\ s\ ]\!] \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda\text{-}.\ Q|\!\},\{\!|\lambda\text{-}.\ E|\!\}$
unfolding *valid-def validE-def*
by (*clarsimp split*: *sum.splits*) *blast*

lemma *validE-valid*: $\{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q|\!\},\{\!|\lambda rv.\ Q|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q|\!\}$
apply (*unfold validE-def*)

apply (*rule hoare-post-imp*)
defer
apply *assumption*
apply (*case-tac r*, *simp-all*)
done

lemma *valid-validE-R*:
$\{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q|\!\},-$
by (*simp add*: *validE-R-def hoare-post-impErr* [*OF valid-validE*])

lemma *valid-validE-E*:
$\{\!|P|\!\}\ f\ \{\!|\lambda rv.\ Q|\!\} \Longrightarrow \{\!|P|\!\}\ f\ -,\{\!|\lambda rv.\ Q|\!\}$
by (*simp add*: *validE-E-def hoare-post-impErr* [*OF valid-validE*])

lemma *validE-validE-R*: $\{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|\top\top|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},-$
by (*simp add*: *validE-R-def*)

lemma *validE-R-validE*: $\{\!|P|\!\}\ f\ \{\!|Q|\!\},- \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|\top\top|\!\}$
by (*simp add*: *validE-R-def*)

lemma *validE-validE-E*: $\{\!|P|\!\}\ f\ \{\!|\top\top|\!\},\{\!|E|\!\} \Longrightarrow \{\!|P|\!\}\ f\ -,\{\!|E|\!\}$
by (*simp add*: *validE-E-def*)

lemma *validE-E-validE*: $\{\!|P|\!\}\ f\ -,\{\!|E|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\top\top|\!\},\{\!|E|\!\}$
by (*simp add*: *validE-E-def*)

lemma *hoare-post-imp-R*: $[\![\ \{\!|P|\!\}\ f\ \{\!|Q'|\!\},-;\ \bigwedge r\ s.\ Q'\ r\ s \Longrightarrow Q\ r\ s\ ]\!] \Longrightarrow \{\!|P|\!\}\ f$
$\{\!|Q|\!\},-$
apply (*unfold validE-R-def*)
apply (*erule hoare-post-impErr*, *simp+*)
done

lemma *hoare-post-imp-E*: $[\![\ \{\!|P|\!\}\ f\ -,\{\!|Q'|\!\};\ \bigwedge r\ s.\ Q'\ r\ s \Longrightarrow Q\ r\ s\ ]\!] \Longrightarrow \{\!|P|\!\}\ f$
$-,\{\!|Q|\!\}$
apply (*unfold validE-E-def*)
apply (*erule hoare-post-impErr*, *simp+*)
done

lemma *hoare-post-comb-imp-conj*:
$[\![\ \{\!|P'|\!\}\ f\ \{\!|Q|\!\};\ \{\!|P|\!\}\ f\ \{\!|Q'|\!\};\ \bigwedge s.\ P\ s \Longrightarrow P'\ s\ ]\!] \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'$
$rv\ s|\!\}$
apply (*rule hoare-pre-imp*)
defer
apply (*rule hoare-vcg-conj-lift*)
apply *assumption+*
apply *simp*
done

lemma *hoare-vcg-precond-impE-R*: $[\![\ \{\!|P'|\!\}\ f\ \{\!|Q|\!\},-;\ \bigwedge s.\ P\ s \Longrightarrow P'\ s\ ]\!] \Longrightarrow \{\!|P|\!\}$

f $\{\!|Q|\!\}$,–
  by (*unfold validE-R-def*, *rule hoare-vcg-precond-impE*, *simp+*)

lemma *valid-is-triple*:
  $valid\ P\ f\ Q = triple\text{-}judgement\ P\ f\ (postcondition\ Q\ (\lambda s\ f.\ fst\ (f\ s)))$
  by (*simp add*: *triple-judgement-def valid-def postcondition-def*)

lemma *validE-is-triple*:
  $validE\ P\ f\ Q\ E = triple\text{-}judgement\ P\ f$
    $(postconditions\ (postcondition\ Q\ (\lambda s\ f.\ \{(rv,\ s').\ (Inr\ rv,\ s') \in fst\ (f\ s)\}))$
        $(postcondition\ E\ (\lambda s\ f.\ \{(rv,\ s').\ (Inl\ rv,\ s') \in fst\ (f\ s)\})))$
  apply (*simp add*: *validE-def triple-judgement-def valid-def postcondition-def*
        postconditions-def split-def split: *sum.split*)
  apply *fastforce*
  done

lemma *validE-R-is-triple*:
  $validE\text{-}R\ P\ f\ Q = triple\text{-}judgement\ P\ f$
    $(postcondition\ Q\ (\lambda s\ f.\ \{(rv,\ s').\ (Inr\ rv,\ s') \in fst\ (f\ s)\}))$
  by (*simp add*: *validE-R-def validE-is-triple postconditions-def postcondition-def*)

lemma *validE-E-is-triple*:
  $validE\text{-}E\ P\ f\ E = triple\text{-}judgement\ P\ f$
    $(postcondition\ E\ (\lambda s\ f.\ \{(rv,\ s').\ (Inl\ rv,\ s') \in fst\ (f\ s)\}))$
  by (*simp add*: *validE-E-def validE-is-triple postconditions-def postcondition-def*)

lemmas *hoare-wp-combs* = *hoare-vcg-conj-lift*

lemmas *hoare-wp-combsE* =
  validE-validE-R
  hoare-vcg-R-conj
  hoare-vcg-E-elim
  hoare-vcg-E-conj

lemmas *hoare-wp-state-combsE* =
  valid-validE-R
  hoare-vcg-R-conj[*OF valid-validE-R*]
  hoare-vcg-E-elim[*OF valid-validE-E*]
  hoare-vcg-E-conj[*OF valid-validE-E*]

lemmas *hoare-classic-wp-combs*
  = *hoare-post-comb-imp-conj hoare-vcg-precond-imp hoare-wp-combs*
lemmas *hoare-classic-wp-combsE*
  = *hoare-vcg-precond-impE hoare-vcg-precond-impE-R hoare-wp-combsE*
lemmas *hoare-classic-wp-state-combsE*
  = *hoare-vcg-precond-impE*[*OF valid-validE*]
  hoare-vcg-precond-impE-R[*OF valid-validE-R*] *hoare-wp-state-combsE*
lemmas *all-classic-wp-combs* =
  hoare-classic-wp-state-combsE hoare-classic-wp-combsE hoare-classic-wp-combs

lemmas *hoare-wp-splits* [*wp-split*] =
hoare-seq-ext hoare-vcg-seqE handleE′-wp handleE-wp
validE-validE-R [*OF hoare-vcg-seqE* [*OF validE-R-validE*]]
validE-validE-R [*OF handleE′-wp* [*OF validE-R-validE*]]
validE-validE-R [*OF handleE-wp* [*OF validE-R-validE*]]
catch-wp hoare-vcg-if-split hoare-vcg-if-splitE
validE-validE-R [*OF hoare-vcg-if-splitE* [*OF validE-R-validE validE-R-validE*]]
liftM-wp liftME-wp
validE-validE-R [*OF liftME-wp* [*OF validE-R-validE*]]
validE-valid

lemmas [*wp-comb*] = *hoare-wp-state-combsE hoare-wp-combsE hoare-wp-combs*

lemmas [*wp*] = *hoare-vcg-prop*
wp-post-taut
return-wp
put-wp
get-wp
gets-wp
modify-wp
returnOk-wp
throwError-wp
fail-wp
failE-wp
liftE-wp
select-f-wp

lemmas [*wp-trip*] = *valid-is-triple validE-is-triple validE-E-is-triple validE-R-is-triple*

lemmas *validE-E-combs*[*wp-comb*] =
hoare-vcg-E-conj[**where** Q'=⊤⊤, *folded validE-E-def*]
valid-validE-E
hoare-vcg-E-conj[**where** Q'=⊤⊤, *folded validE-E-def*, *OF valid-validE-E*]

Simplifications on conjunction

lemma *hoare-post-eq*: $\llbracket\ Q = Q';\ \{\!|P|\!\}\ f\ \{\!|Q'|\!\}\ \rrbracket \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\}$
by *simp*
lemma *hoare-post-eqE1*: $\llbracket\ Q = Q';\ \{\!|P|\!\}\ f\ \{\!|Q'|\!\},\{\!|E|\!\}\ \rrbracket \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\}$
by *simp*
lemma *hoare-post-eqE2*: $\llbracket\ E = E';\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|E'|\!\}\ \rrbracket \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\}$
by *simp*
lemma *hoare-post-eqE-R*: $\llbracket\ Q = Q';\ \{\!|P|\!\}\ f\ \{\!|Q'|\!\},-\ \rrbracket \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},-$
by *simp*

lemma *pred-conj-apply-elim*: $(\lambda r.\ Q\ r\ \mathit{and}\ Q'\ r) = (\lambda r\ s.\ Q\ r\ s \wedge Q'\ r\ s)$
by (*simp add*: *pred-conj-def*)
lemma *pred-conj-conj-elim*: $(\lambda r\ s.\ (Q\ r\ \mathit{and}\ Q'\ r)\ s \wedge Q''\ r\ s) = (\lambda r\ s.\ Q\ r\ s \wedge Q'\ r\ s \wedge Q''\ r\ s)$

by *simp*
lemma *conj-assoc-apply*: $(\lambda r\ s.\ (Q\ r\ s \wedge Q'\ r\ s) \wedge Q''\ r\ s) = (\lambda r\ s.\ Q\ r\ s \wedge Q'\ r\ s \wedge Q''\ r\ s)$
by *simp*
lemma *all-elim*: $(\lambda rv\ s.\ \forall x.\ P\ rv\ s) = P$
by *simp*
lemma *all-conj-elim*: $(\lambda rv\ s.\ (\forall x.\ P\ rv\ s) \wedge Q\ rv\ s) = (\lambda rv\ s.\ P\ rv\ s \wedge Q\ rv\ s)$
by *simp*

lemmas *vcg-rhs-simps* = *pred-conj-apply-elim pred-conj-conj-elim conj-assoc-apply all-elim all-conj-elim*

lemma *if-apply-reduct*: $\{\!|P|\!\}$ *If* P' $(f\ x)$ $(g\ x)$ $\{\!|Q|\!\} \Longrightarrow \{\!|P|\!\}$ *If* $P'\ f\ g\ x$ $\{\!|Q|\!\}$
by (*cases* P', *simp-all*)
lemma *if-apply-reductE*: $\{\!|P|\!\}$ *If* P' $(f\ x)$ $(g\ x)$ $\{\!|Q|\!\},\{\!|E|\!\} \Longrightarrow \{\!|P|\!\}$ *If* $P'\ f\ g\ x$ $\{\!|Q|\!\},\{\!|E|\!\}$
by (*cases* P', *simp-all*)
lemma *if-apply-reductE-R*: $\{\!|P|\!\}$ *If* P' $(f\ x)$ $(g\ x)$ $\{\!|Q|\!\},- \Longrightarrow \{\!|P|\!\}$ *If* $P'\ f\ g\ x$ $\{\!|Q|\!\},-$
by (*cases* P', *simp-all*)

lemmas *hoare-wp-simps* [*wp-split*] =
vcg-rhs-simps [*THEN hoare-post-eq*] *vcg-rhs-simps* [*THEN hoare-post-eqE1*]
vcg-rhs-simps [*THEN hoare-post-eqE2*] *vcg-rhs-simps* [*THEN hoare-post-eqE-R*]
if-apply-reduct if-apply-reductE if-apply-reductE-R TrueI

schematic-goal *if-apply-test*: $\{\!|?Q|\!\}$ (*if A then returnOk else K fail*) x $\{\!|P|\!\},\{\!|E|\!\}$
by *wpsimp*

lemma *hoare-elim-pred-conj*:
$\{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ r\ s \wedge Q'\ r\ s|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r.\ Q\ r$ *and* $Q'\ r|\!\}$
by (*unfold pred-conj-def*)

lemma *hoare-elim-pred-conjE1*:
$\{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ r\ s \wedge Q'\ r\ s|\!\},\{\!|E|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r.\ Q\ r$ *and* $Q'\ r|\!\},\{\!|E|\!\}$
by (*unfold pred-conj-def*)

lemma *hoare-elim-pred-conjE2*:
$\{\!|P|\!\}\ f\ \{\!|Q|\!\},\ \{\!|\lambda x\ s.\ E\ x\ s \wedge E'\ x\ s|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|\lambda x.\ E\ x$ *and* $E'\ x|\!\}$
by (*unfold pred-conj-def*)

lemma *hoare-elim-pred-conjE-R*:
$\{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ r\ s \wedge Q'\ r\ s|\!\},- \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r.\ Q\ r$ *and* $Q'\ r|\!\},-$
by (*unfold pred-conj-def*)

lemmas *hoare-wp-pred-conj-elims* =
hoare-elim-pred-conj hoare-elim-pred-conjE1
hoare-elim-pred-conjE2 hoare-elim-pred-conjE-R

lemmas *hoare-weaken-preE* = *hoare-vcg-precond-impE*

lemmas *hoare-pre* [*wp-pre*] =
hoare-weaken-pre
hoare-weaken-preE
hoare-vcg-precond-impE-R
hoare-weaken-preE-E

declare *no-fail-pre* [*wp-pre*]

bundle *no-pre* = *hoare-pre* [*wp-pre del*] *no-fail-pre* [*wp-pre del*]

bundle *classic-wp-pre* = *hoare-pre* [*wp-pre del*] *no-fail-pre* [*wp-pre del*]
all-classic-wp-combs[*wp-comb del*] *all-classic-wp-combs*[*wp-comb*]

Miscellaneous lemmas on hoare triples

lemma *hoare-vcg-mp*:
assumes *a*: $\{\!|P|\!\}\ f\ \{\!|Q|\!\}$
assumes *b*: $\{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ r\ s \longrightarrow Q'\ r\ s|\!\}$
shows $\{\!|P|\!\}\ f\ \{\!|Q'|\!\}$
using *assms*
by (*auto simp*: *valid-def split-def*)

lemma *hoare-add-post*:
assumes *r*: $\{\!|P'|\!\}\ f\ \{\!|Q'|\!\}$
assumes *impP*: $\bigwedge s.\ P\ s \Longrightarrow P'\ s$
assumes *impQ*: $\{\!|P|\!\}\ f\ \{\!|\lambda rv\ s.\ Q'\ rv\ s \longrightarrow Q\ rv\ s|\!\}$
shows $\{\!|P|\!\}\ f\ \{\!|Q|\!\}$
apply (*rule hoare-chain*)
apply (*rule hoare-vcg-conj-lift*)
apply (*rule r*)
apply (*rule impQ*)
apply *simp*
apply (*erule impP*)
apply *simp*
done

lemma *hoare-gen-asmE*:
$(P \Longrightarrow \{\!|P'|\!\}\ f\ \{\!|Q|\!\},-) \Longrightarrow \{\!|P'\ and\ K\ P|\!\}\ f\ \{\!|Q|\!\}, -$
by (*simp add*: *validE-R-def validE-def valid-def*) *blast*

lemma *hoare-list-case*:
assumes *P1*: $\{\!|P1|\!\}\ f\ f1\ \{\!|Q|\!\}$
assumes *P2*: $\bigwedge y\ ys.\ xs = y\#ys \Longrightarrow \{\!|P2\ y\ ys|\!\}\ f\ (f2\ y\ ys)\ \{\!|Q|\!\}$
shows $\{\!|case\ xs\ of\ [] \Rightarrow P1 \mid y\#ys \Rightarrow P2\ y\ ys|\!\}$
$f\ (case\ xs\ of\ [] \Rightarrow f1 \mid y\#ys \Rightarrow f2\ y\ ys)$
$\{\!|Q|\!\}$
apply (*cases xs*; *simp*)

apply (*rule P1*)
apply (*rule P2*)
apply *simp*
done

lemma *hoare-when-wp* [*wp-split*]:
$\llbracket$ $P \Longrightarrow \{\!|Q|\!\}\ f\ \{\!|R|\!\}$ $\rrbracket \Longrightarrow \{\!|$*if P then Q else R* ()$|\!\}$ *when P f* $\{\!|R|\!\}$
by (*clarsimp simp*: *when-def valid-def return-def*)

lemma *hoare-unless-wp*[*wp-split*]:
$(\neg P \Longrightarrow \{\!|Q|\!\}\ f\ \{\!|R|\!\}) \Longrightarrow \{\!|$*if P then R* () *else Q*$|\!\}$ *unless P f* $\{\!|R|\!\}$
unfolding *unless-def* **by** *wp auto*

lemma *hoare-whenE-wp*:
$(P \Longrightarrow \{\!|Q|\!\}\ f\ \{\!|R|\!\}, \{\!|E|\!\}) \Longrightarrow \{\!|$*if P then Q else R* ()$|\!\}$ *whenE P f* $\{\!|R|\!\}, \{\!|E|\!\}$
unfolding *whenE-def* **by** *clarsimp wp*

lemmas *hoare-whenE-wps*[*wp-split*]
= *hoare-whenE-wp hoare-whenE-wp*[*THEN validE-validE-R*] *hoare-whenE-wp*[*THEN validE-validE-E*]

lemma *hoare-unlessE-wp*:
$(\neg\ P \Longrightarrow \{\!|Q|\!\}\ f\ \{\!|R|\!\}, \{\!|E|\!\}) \Longrightarrow \{\!|$*if P then R* () *else Q*$|\!\}$ *unlessE P f* $\{\!|R|\!\}, \{\!|E|\!\}$
unfolding *unlessE-def* **by** *wp auto*

lemmas *hoare-unlessE-wps*[*wp-split*]
= *hoare-unlessE-wp hoare-unlessE-wp*[*THEN validE-validE-R*] *hoare-unlessE-wp*[*THEN validE-validE-E*]

lemma *hoare-use-eq*:
assumes x: $\bigwedge P.\ \{\!|\lambda s.\ P\ (f\ s)|\!\}\ m\ \{\!|\lambda rv\ s.\ P\ (f\ s)|\!\}$
assumes y: $\bigwedge f.\ \{\!|\lambda s.\ P\ f\ s|\!\}\ m\ \{\!|\lambda rv\ s.\ Q\ f\ s|\!\}$
shows $\{\!|\lambda s.\ P\ (f\ s)\ s|\!\}\ m\ \{\!|\lambda rv\ s.\ Q\ (f\ s :: {'}c :: type)\ s\ |\!\}$
apply (*rule-tac* $Q{=}\lambda rv\ s.\ \exists f'.\ f' = f\ s \wedge Q\ f'\ s$ **in** *hoare-post-imp*)
apply *simp*
apply (*wpsimp wp*: *hoare-vcg-ex-lift x y*)
done

lemma *hoare-return-sp*:
$\{\!|P|\!\}$ *return x* $\{\!|\lambda r.\ P$ *and* $K\ (r = x)|\!\}$
by (*simp add*: *valid-def return-def*)

lemma *hoare-fail-any* [*simp*]:
$\{\!|P|\!\}$ *fail* $\{\!|Q|\!\}$ **by** *wp*

lemma *hoare-failE* [*simp*]: $\{\!|P|\!\}$ *fail* $\{\!|Q|\!\},\{\!|E|\!\}$ **by** *wp*

lemma *hoare-FalseE* [*simp*]:
$\{\!|\lambda s.$ *False*$|\!\}\ f\ \{\!|Q|\!\},\{\!|E|\!\}$

by (*simp add*: *valid-def validE-def*)

lemma *hoare-K-bind* [*wp-split*]:
$\{\!|P|\!\}\ f\ \{\!|Q|\!\} \Longrightarrow \{\!|P|\!\}\ \textit{K-bind}\ f\ x\ \{\!|Q|\!\}$
by *simp*

lemma *validE-K-bind* [*wp-split*]:
$\{\!|\ P\ |\!\}\ x\ \{\!|\ Q\ |\!\},\ \{\!|\ E\ |\!\} \Longrightarrow \{\!|\ P\ |\!\}\ \textit{K-bind}\ x\ f\ \{\!|\ Q\ |\!\},\ \{\!|\ E\ |\!\}$
by *simp*

Setting up the precondition case splitter.

lemma *wpc-helper-valid*:
$\{\!|Q|\!\}\ g\ \{\!|S|\!\} \Longrightarrow \textit{wpc-helper}\ (P,\ P')\ (Q,\ Q')\ \{\!|P|\!\}\ g\ \{\!|S|\!\}$
by (*clarsimp simp*: *wpc-helper-def elim*!: *hoare-pre*)

lemma *wpc-helper-validE*:
$\{\!|Q|\!\}\ f\ \{\!|R|\!\},\{\!|E|\!\} \Longrightarrow \textit{wpc-helper}\ (P,\ P')\ (Q,\ Q')\ \{\!|P|\!\}\ f\ \{\!|R|\!\},\{\!|E|\!\}$
by (*clarsimp simp*: *wpc-helper-def elim*!: *hoare-pre*)

lemma *wpc-helper-validE-R*:
$\{\!|Q|\!\}\ f\ \{\!|R|\!\},- \Longrightarrow \textit{wpc-helper}\ (P,\ P')\ (Q,\ Q')\ \{\!|P|\!\}\ f\ \{\!|R|\!\},-$
by (*clarsimp simp*: *wpc-helper-def elim*!: *hoare-pre*)

lemma *wpc-helper-validR-R*:
$\{\!|Q|\!\}\ f\ -,\{\!|E|\!\} \Longrightarrow \textit{wpc-helper}\ (P,\ P')\ (Q,\ Q')\ \{\!|P|\!\}\ f\ -,\{\!|E|\!\}$
by (*clarsimp simp*: *wpc-helper-def elim*!: *hoare-pre*)

lemma *wpc-helper-no-fail-final*:
$\textit{no-fail}\ Q\ f \Longrightarrow \textit{wpc-helper}\ (P,\ P')\ (Q,\ Q')\ (\textit{no-fail}\ P\ f)$
by (*clarsimp simp*: *wpc-helper-def elim*!: *no-fail-pre*)

lemma *wpc-helper-empty-fail-final*:
$\textit{empty-fail}\ f \Longrightarrow \textit{wpc-helper}\ (P,\ P')\ (Q,\ Q')\ (\textit{empty-fail}\ f)$
by (*clarsimp simp*: *wpc-helper-def*)

lemma *wpc-helper-validNF*:
$\{\!|Q|\!\}\ g\ \{\!|S|\!\}! \Longrightarrow \textit{wpc-helper}\ (P,\ P')\ (Q,\ Q')\ \{\!|P|\!\}\ g\ \{\!|S|\!\}!$
apply (*clarsimp simp*: *wpc-helper-def*)
by (*metis hoare-vcg-precond-imp no-fail-pre validNF-def*)

wpc-setup $\lambda m.\ \{\!|P|\!\}\ m\ \{\!|Q|\!\}$ *wpc-helper-valid*
wpc-setup $\lambda m.\ \{\!|P|\!\}\ m\ \{\!|Q|\!\},\{\!|E|\!\}$ *wpc-helper-validE*
wpc-setup $\lambda m.\ \{\!|P|\!\}\ m\ \{\!|Q|\!\},-$ *wpc-helper-validE-R*
wpc-setup $\lambda m.\ \{\!|P|\!\}\ m\ -,\{\!|E|\!\}$ *wpc-helper-validR-R*
wpc-setup $\lambda m.$ *no-fail P m wpc-helper-no-fail-final*
wpc-setup $\lambda m.$ *empty-fail m wpc-helper-empty-fail-final*
wpc-setup $\lambda m.\ \{\!|P|\!\}\ m\ \{\!|Q|\!\}!$ *wpc-helper-validNF*

lemma *in-liftM*:

$((r,\ s') \in \mathit{fst}\ (\mathit{liftM}\ t\ f\ s)) = (\exists\, r'.\ (r',\ s') \in \mathit{fst}\ (f\ s) \wedge r = t\ r')$
apply (*simp add*: *liftM-def return-def bind-def*)
apply (*simp add*: *Bex-def*)
done

lemmas *handy-liftM-lemma* = *in-liftM*

lemma *hoare-fun-app-wp*[*wp*]:
$\{\!|P|\!\}\ f'\ x\ \{\!|Q'|\!\} \Longrightarrow \{\!|P|\!\}\ f'\ \$\ x\ \{\!|Q'|\!\}$
$\{\!|P|\!\}\ f\,x\ \{\!|Q|\!\},\{\!|E|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \$\ x\ \{\!|Q|\!\},\{\!|E|\!\}$
$\{\!|P|\!\}\ f\,x\ \{\!|Q|\!\},- \Longrightarrow \{\!|P|\!\}\ f\ \$\ x\ \{\!|Q|\!\},-$
$\{\!|P|\!\}\ f\,x\ -,\{\!|E|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \$\ x\ -,\{\!|E|\!\}$
by *simp*+

lemma *hoare-validE-pred-conj*:
$[\![\ \{\!|P|\!\}f\{\!|Q|\!\},\{\!|E|\!\};\ \{\!|P|\!\}f\{\!|R|\!\},\{\!|E|\!\}\ ]\!] \Longrightarrow \{\!|P|\!\}f\{\!|Q\ \mathit{And}\ R|\!\},\{\!|E|\!\}$
unfolding *valid-def validE-def* **by** (*simp add*: *split-def split*: *sum.splits*)

lemma *hoare-validE-conj*:
$[\![\ \{\!|P|\!\}f\{\!|Q|\!\},\{\!|E|\!\};\ \{\!|P|\!\}f\{\!|R|\!\},\{\!|E|\!\}\ ]\!] \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ Q\ r\ s \wedge R\ r\ s|\!\},\{\!|E|\!\}$
unfolding *valid-def validE-def* **by** (*simp add*: *split-def split*: *sum.splits*)

lemmas *hoare-valid-validE* = *valid-validE*

lemma *liftE-validE-E* [*wp*]:
$\{\!|\top|\!\}\ \mathit{liftE}\ f\ -,\ \{\!|Q|\!\}$
by (*clarsimp simp*: *validE-E-def valid-def*)

declare *validE-validE-E*[*wp-comb*]

lemmas *if-validE-E* [*wp-split*] =
validE-validE-E [*OF hoare-vcg-if-splitE* [*OF validE-E-validE validE-E-validE*]]

lemma *returnOk-E* [*wp*]:
$\{\!|\top|\!\}\ \mathit{returnOk}\ r\ -,\ \{\!|Q|\!\}$
by (*simp add*: *validE-E-def*) *wp*

lemma *hoare-drop-imp*:
$\{\!|P|\!\}\ f\ \{\!|Q|\!\} \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ R\ r\ s \longrightarrow Q\ r\ s|\!\}$
by (*auto simp*: *valid-def*)

lemma *hoare-drop-impE*:
$[\![\{\!|P|\!\}\ f\ \{\!|\lambda r.\ Q|\!\},\ \{\!|E|\!\}]\!] \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ R\ r\ s \longrightarrow Q\ s|\!\},\ \{\!|E|\!\}$
by (*simp add*: *validE-weaken*)

lemma *hoare-drop-impE-R*:
$\{\!|P|\!\}\ f\ \{\!|Q|\!\},- \Longrightarrow \{\!|P|\!\}\ f\ \{\!|\lambda r\ s.\ R\ r\ s \longrightarrow Q\ r\ s|\!\},\ -$

by (*auto simp*: *validE-R-def validE-def valid-def split-def split*: *sum.splits*)

lemma *hoare-drop-impE-E*:
$\{\!|P|\!\}\ f\ -,\{\!|Q|\!\} \Longrightarrow \{\!|P|\!\}\ f\ -,\{\!|\lambda r\ s.\ R\ r\ s \longrightarrow Q\ r\ s|\!\}$
by (*auto simp*: *validE-E-def validE-def valid-def split-def split*: *sum.splits*)

lemmas *hoare-drop-imps* = *hoare-drop-imp hoare-drop-impE-R hoare-drop-impE-E*

lemma *hoare-drop-imp-conj*[*wp-unsafe*]:
$\{\!|P|\!\}\ f\ \{\!|Q'|\!\} \Longrightarrow \{\!|P'|\!\}\ f\ \{\!|\lambda rv\ s.\ (Q\ rv\ s \longrightarrow Q''\ rv\ s) \wedge Q'''\ rv\ s|\!\}$
$\Longrightarrow \{\!|P\ and\ P'|\!\}\ f\ \{\!|\lambda rv\ s.\ (Q\ rv\ s \longrightarrow Q'\ rv\ s \wedge Q''\ rv\ s) \wedge Q'''\ rv\ s|\!\}$
by (*auto simp*: *valid-def*)

lemmas *hoare-drop-imp-conj'*[*wp-unsafe*] = *hoare-drop-imp-conj*[**where** Q'''=⊤⊤, *simplified*]

lemma *bind-det-exec*:
$fst\ (a\ s) = \{(r,s')\} \Longrightarrow fst\ ((a >>= b)\ s) = fst\ (b\ r\ s')$
by (*simp add*: *bind-def*)

lemma *in-bind-det-exec*:
$fst\ (a\ s) = \{(r,s')\} \Longrightarrow (s'' \in fst\ ((a >>= b)\ s)) = (s'' \in fst\ (b\ r\ s'))$
by (*simp add*: *bind-def*)

lemma *exec-put*:
$(put\ s' >>= m)\ s = m\ ()\ s'$
by (*simp add*: *bind-def put-def*)

lemma *bind-execI*:
$[\![\ (r'',s'') \in fst\ (f\ s);\ \exists\, x \in fst\ (g\ r''\ s'').\ P\ x\ ]\!] \Longrightarrow$
$\exists\, x \in fst\ ((f >>= g)\ s).\ P\ x$
by (*force simp*: *in-bind split-def bind-def*)

lemma *True-E-E* [*wp*]: $\{\!|\top|\!\}\ f\ -,\{\!|\top\top|\!\}$
by (*auto simp*: *validE-E-def validE-def valid-def split*: *sum.splits*)

lemmas [*wp-split*] =
validE-validE-E [*OF hoare-vcg-seqE* [*OF validE-E-validE*]]

lemma *case-option-wp*:
assumes *x*: $\bigwedge x.\ \{\!|P\ x|\!\}\ m\ x\ \{\!|Q|\!\}$
assumes *y*: $\{\!|P'|\!\}\ m'\ \{\!|Q|\!\}$
shows $\{\!|\lambda s.\ (x = None \longrightarrow P'\ s) \wedge (x \neq None \longrightarrow P\ (the\ x)\ s)|\!\}$
$case\text{-}option\ m'\ m\ x\ \{\!|Q|\!\}$
apply (*cases x*; *simp*)
apply (*rule y*)
apply (*rule x*)

done

lemma *case-option-wpE*:
assumes x: $\bigwedge x.\ \{\!|P\ x|\!\}\ m\ x\ \{\!|Q|\!\},\{\!|E|\!\}$
assumes y: $\{\!|P'|\!\}\ m'\ \{\!|Q|\!\},\{\!|E|\!\}$
shows $\{\!|\lambda s.\ (x = None \longrightarrow P'\ s) \wedge (x \neq None \longrightarrow P\ (the\ x)\ s)|\!\}$
$case\text{-}option\ m'\ m\ x\ \{\!|Q|\!\},\{\!|E|\!\}$
apply (*cases x*; *simp*)
apply (*rule y*)
apply (*rule x*)
done

lemma *in-bindE*:
$(rv, s') \in fst\ ((f >>=E (\lambda rv'.\ g\ rv'))\ s) =$
$((\exists ex.\ rv = Inl\ ex \wedge (Inl\ ex, s') \in fst\ (f\ s)) \vee$
$(\exists rv'\ s''.\ (rv, s') \in fst\ (g\ rv'\ s'') \wedge (Inr\ rv', s'') \in fst\ (f\ s)))$
apply (*rule iffI*)
apply (*clarsimp simp*: *bindE-def bind-def*)
apply (*case-tac a*)
apply (*clarsimp simp*: *lift-def throwError-def return-def*)
apply (*clarsimp simp*: *lift-def*)
apply *safe*
apply (*clarsimp simp*: *bindE-def bind-def*)
apply (*erule rev-bexI*)
apply (*simp add*: *lift-def throwError-def return-def*)
apply (*clarsimp simp*: *bindE-def bind-def*)
apply (*erule rev-bexI*)
apply (*simp add*: *lift-def*)
done

lemmas [*wp-split*] = *validE-validE-E* [*OF liftME-wp*, *simplified*, *OF validE-E-validE*]

lemma *assert-A-True*[*simp*]: *assert True* = *return* ()
by (*simp add*: *assert-def*)

lemma *assert-wp* [*wp*]: $\{\!|\lambda s.\ P \longrightarrow Q\ ()\ s|\!\}\ assert\ P\ \{\!|Q|\!\}$
by (*cases P*, (*simp add*: *assert-def* | *wp*)+)

lemma *list-cases-wp*:
assumes a: $\{\!|P\text{-}A|\!\}\ a\ \{\!|Q|\!\}$
assumes b: $\bigwedge x\ xs.\ ts = x\#xs \Longrightarrow \{\!|P\text{-}B\ x\ xs|\!\}\ b\ x\ xs\ \{\!|Q|\!\}$
shows $\{\!|case\text{-}list\ P\text{-}A\ P\text{-}B\ ts|\!\}\ case\ ts\ of\ [] \Rightarrow a \mid x\ \#\ xs \Rightarrow b\ x\ xs\ \{\!|Q|\!\}$
by (*cases ts*, *auto simp*: *a b*)

lemma *whenE-throwError-wp*:
$\{\!|\lambda s.\ \neg Q \longrightarrow P\ s|\!\}\ whenE\ Q\ (throwError\ e)\ \{\!|\lambda rv.\ P|\!\}, -$
unfolding *whenE-def* **by** *wpsimp*

lemma *select-throwError-wp*:
$\{\!|\lambda s.\ \forall x{\in}S.\ Q\ x\ s|\!\}$ *select* S >>= *throwError* $-$, $\{\!|Q|\!\}$
by (*simp add*: *bind-def throwError-def return-def select-def validE-E-def validE-def valid-def*)

lemma *assert-opt-wp*[*wp*]:
$\{\!|\lambda s.\ x \neq None \longrightarrow Q\ (the\ x)\ s|\!\}$ *assert-opt* x $\{\!|Q|\!\}$
by (*case-tac* x, (*simp add*: *assert-opt-def* | *wp*)+)

lemma *gets-the-wp*[*wp*]:
$\{\!|\lambda s.\ (f\ s \neq None) \longrightarrow Q\ (the\ (f\ s))\ s|\!\}$ *gets-the* f $\{\!|Q|\!\}$
by (*unfold gets-the-def*, *wp*)

lemma *gets-the-wp'*:
$\{\!|\lambda s.\ \forall rv.\ f\ s = Some\ rv \longrightarrow Q\ rv\ s|\!\}$ *gets-the* f $\{\!|Q|\!\}$
unfolding *gets-the-def* **by** *wpsimp*

lemma *gets-map-wp*:
$\{\!|\lambda s.\ f\ s\ p \neq None \longrightarrow Q\ (the\ (f\ s\ p))\ s|\!\}$ *gets-map* $f\ p$ $\{\!|Q|\!\}$
unfolding *gets-map-def* **by** *wpsimp*

lemma *gets-map-wp'*[*wp*]:
$\{\!|\lambda s.\ \forall rv.\ f\ s\ p = Some\ rv \longrightarrow Q\ rv\ s|\!\}$ *gets-map* $f\ p$ $\{\!|Q|\!\}$
unfolding *gets-map-def* **by** *wpsimp*

lemma *no-fail-gets-map*[*wp*]:
no-fail $(\lambda s.\ f\ s\ p \neq None)$ (*gets-map* $f\ p$)
unfolding *gets-map-def* **by** *wpsimp*

lemma *hoare-vcg-set-pred-lift*:
assumes $\bigwedge P\ x.\ m\ \{\!|\ \lambda s.\ P\ (f\ x\ s)\ |\!\}$
shows $m\ \{\!|\ \lambda s.\ P\ \{x.\ f\ x\ s\}\ |\!\}$
using *assms*[**where** $P{=}\lambda x\ .\ x$] *assms*[**where** $P{=}Not$] *use-valid*
by (*fastforce simp*: *valid-def elim!*: *rsubst*[**where** $P{=}P$])

lemma *hoare-vcg-set-pred-lift-mono*:
assumes f: $\bigwedge x.\ m\ \{\!|\ f\ x\ |\!\}$
assumes *mono*: $\bigwedge A\ B.\ A \subseteq B \Longrightarrow P\ A \Longrightarrow P\ B$
shows $m\ \{\!|\ \lambda s.\ P\ \{x.\ f\ x\ s\}\ |\!\}$
by (*fastforce simp*: *valid-def elim!*: *mono*[*rotated*] *dest*: *use-valid*[*OF* - f])

24 validNF Rules

24.1 Basic validNF theorems

lemma *validNF* [*intro?*]:
$[\![\ \{\!|\ P\ |\!\}\ f\ \{\!|\ Q\ |\!\};\ no\text{-}fail\ P\ f\ ]\!] \Longrightarrow \{\!|\ P\ |\!\}\ f\ \{\!|\ Q\ |\!\}!$
by (*clarsimp simp*: *validNF-def*)

lemma *validNF-valid*: $\llbracket$ $\{\!|$ P $|\!\}$ f $\{\!|$ Q $|\!\}!$ $\rrbracket \Longrightarrow$ $\{\!|$ P $|\!\}$ f $\{\!|$ Q $|\!\}$
by (*clarsimp simp*: *validNF-def*)

lemma *validNF-no-fail*: $\llbracket$ $\{\!|$ P $|\!\}$ f $\{\!|$ Q $|\!\}!$ $\rrbracket \Longrightarrow$ *no-fail P f*
by (*clarsimp simp*: *validNF-def*)

lemma *snd-validNF*:
$\llbracket$ $\{\!|$ P $|\!\}$ f $\{\!|$ Q $|\!\}!$; $P\ s$ $\rrbracket \Longrightarrow \neg$ *snd* $(f\ s)$
by (*clarsimp simp*: *validNF-def no-fail-def*)

lemma *use-validNF*:
$\llbracket$ $(r', s') \in$ *fst* $(f\ s)$; $\{\!|$ P $|\!\}$ f $\{\!|$ Q $|\!\}!$; $P\ s$ $\rrbracket \Longrightarrow Q\ r'\ s'$
by (*fastforce simp*: *validNF-def valid-def*)

24.2 validNF weakest pre-condition rules

lemma *validNF-return* [*wp*]:
$\{\!|$ $P\ x$ $|\!\}$ *return* x $\{\!|$ P $|\!\}!$
by (*wp validNF*)+

lemma *validNF-get* [*wp*]:
$\{\!|$ $\lambda s.\ P\ s\ s$ $|\!\}$ *get* $\{\!|$ P $|\!\}!$
by (*wp validNF*)+

lemma *validNF-put* [*wp*]:
$\{\!|$ $\lambda s.\ P\ ()\ x$ $|\!\}$ *put* x $\{\!|$ P $|\!\}!$
by (*wp validNF*)+

lemma *validNF-K-bind* [*wp*]:
$\{\!|$ P $|\!\}$ x $\{\!|$ Q $|\!\}! \Longrightarrow$ $\{\!|$ P $|\!\}$ *K-bind x f* $\{\!|$ Q $|\!\}!$
by *simp*

lemma *validNF-fail* [*wp*]:
$\{\!|$ $\lambda s.$ *False* $|\!\}$ *fail* $\{\!|$ Q $|\!\}!$
by (*clarsimp simp*: *validNF-def fail-def no-fail-def*)

lemma *validNF-prop* [*wp-unsafe*]:
$\llbracket$ *no-fail* $(\lambda s.\ P)$ f $\rrbracket \Longrightarrow$ $\{\!|$ $\lambda s.\ P$ $|\!\}$ f $\{\!|$ $\lambda rv\ s.\ P$ $|\!\}!$
by (*wp validNF*)+

lemma *validNF-post-conj* [*intro*!]:
$\llbracket$ $\{\!|$ P $|\!\}$ a $\{\!|$ Q $|\!\}!$; $\{\!|$ P $|\!\}$ a $\{\!|$ R $|\!\}!$ $\rrbracket \Longrightarrow$ $\{\!|$ P $|\!\}$ a $\{\!|$ Q *And* R $|\!\}!$
by (*auto simp*: *validNF-def*)

lemma *no-fail-or*:
$\llbracket$*no-fail* $P\ a$; *no-fail* $Q\ a\rrbracket \Longrightarrow$ *no-fail* $(P$ *or* $Q)$ a
by (*clarsimp simp*: *no-fail-def*)

lemma *validNF-pre-disj* [*intro*!]:
$\llbracket$ $\{\!|$ P $|\!\}$ a $\{\!|$ R $|\!\}$!; $\{\!|$ Q $|\!\}$ a $\{\!|$ R $|\!\}$! $\rrbracket \Longrightarrow \{\!|$ P *or* Q $|\!\}$ a $\{\!|$ R $|\!\}$!
by (*rule validNF*) (*auto dest*: *validNF-valid validNF-no-fail intro*: *no-fail-or*)

definition *validNF-property* Q s $b \equiv \neg$ *snd* $(b\ s) \wedge (\forall (r', s') \in$ *fst* $(b\ s).\ Q\ r'\ s')$

lemma *validNF-is-triple* [*wp-trip*]:
validNF P f Q = *triple-judgement* P f (*validNF-property* Q)
apply (*clarsimp simp*: *validNF-def triple-judgement-def validNF-property-def*)
apply (*auto simp*: *no-fail-def valid-def*)
done

lemma *validNF-weaken-pre*[*wp-pre*]:
$\llbracket\{\!|Q|\!\}$ a $\{\!|R|\!\}$!; $\bigwedge s.\ P\ s \Longrightarrow Q\ s\rrbracket \Longrightarrow \{\!|P|\!\}$ a $\{\!|R|\!\}$!
by (*metis hoare-pre-imp no-fail-pre validNF-def*)

lemma *validNF-post-comb-imp-conj*:
$\llbracket$ $\{\!|P'|\!\}$ f $\{\!|Q|\!\}$!; $\{\!|P|\!\}$ f $\{\!|Q'|\!\}$!; $\bigwedge s.\ P\ s \Longrightarrow P'\ s$ $\rrbracket \Longrightarrow \{\!|P|\!\}$ f $\{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'$ $rv\ s|\!\}$!
by (*fastforce simp*: *validNF-def valid-def*)

lemma *validNF-post-comb-conj-L*:
$\llbracket$ $\{\!|P'|\!\}$ f $\{\!|Q|\!\}$!; $\{\!|P|\!\}$ f $\{\!|Q'|\!\}$ $\rrbracket \Longrightarrow \{\!|\lambda s.\ P\ s \wedge P'\ s$ $|\!\}$ f $\{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s|\!\}$!
apply (*clarsimp simp*: *validNF-def valid-def no-fail-def*)
apply *force*
done

lemma *validNF-post-comb-conj-R*:
$\llbracket$ $\{\!|P'|\!\}$ f $\{\!|Q|\!\}$; $\{\!|P|\!\}$ f $\{\!|Q'|\!\}$! $\rrbracket \Longrightarrow \{\!|\lambda s.\ P\ s \wedge P'\ s$ $|\!\}$ f $\{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s|\!\}$!
apply (*clarsimp simp*: *validNF-def valid-def no-fail-def*)
apply *force*
done

lemma *validNF-post-comb-conj*:
$\llbracket$ $\{\!|P'|\!\}$ f $\{\!|Q|\!\}$!; $\{\!|P|\!\}$ f $\{\!|Q'|\!\}$! $\rrbracket \Longrightarrow \{\!|\lambda s.\ P\ s \wedge P'\ s$ $|\!\}$ f $\{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv$ $s|\!\}$!
apply (*clarsimp simp*: *validNF-def valid-def no-fail-def*)
apply *force*
done

lemma *validNF-if-split* [*wp-split*]:
$\llbracket P \Longrightarrow \{\!|Q|\!\}$ f $\{\!|S|\!\}$!; $\neg P \Longrightarrow \{\!|R|\!\}$ g $\{\!|S|\!\}$!$\rrbracket \Longrightarrow \{\!|\lambda s.\ (P \longrightarrow Q\ s) \wedge (\neg P \longrightarrow R$ $s)|\!\}$ *if* P *then* f *else* g $\{\!|S|\!\}$!
by *simp*

lemma *validNF-vcg-conj-lift*:
$\llbracket$ $\{\!|P|\!\}$ f $\{\!|Q|\!\}$!; $\{\!|P'|\!\}$ f $\{\!|Q'|\!\}$! $\rrbracket \Longrightarrow$

$\{\!|\lambda s.\ P\ s \wedge P'\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s|\!\}!$
apply (*subst bipred-conj-def*[*symmetric*], *rule validNF-post-conj*)
apply (*erule validNF-weaken-pre*, *fastforce*)
apply (*erule validNF-weaken-pre*, *fastforce*)
done

lemma *validNF-vcg-disj-lift*:
$[\![\ \{\!|P|\!\}\ f\ \{\!|Q|\!\}!;\ \{\!|P'|\!\}\ f\ \{\!|Q'|\!\}!\ ]\!] \Longrightarrow$
$\{\!|\lambda s.\ P\ s \vee P'\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ Q\ rv\ s \vee Q'\ rv\ s|\!\}!$
apply (*clarsimp simp*: *validNF-def*)
apply *safe*
apply (*auto intro*!: *hoare-vcg-disj-lift*)[*1*]
apply (*clarsimp simp*: *no-fail-def*)
done

lemma *validNF-vcg-all-lift* [*wp*]:
$[\![\ \bigwedge x.\ \{\!|P\ x|\!\}\ f\ \{\!|Q\ x|\!\}!\ ]\!] \Longrightarrow \{\!|\lambda s.\ \forall x.\ P\ x\ s|\!\}\ f\ \{\!|\lambda rv\ s.\ \forall x.\ Q\ x\ rv\ s|\!\}!$
apply *atomize*
apply (*rule validNF*)
apply (*clarsimp simp*: *validNF-def*)
apply (*rule hoare-vcg-all-lift*)
apply *force*
apply (*clarsimp simp*: *no-fail-def validNF-def*)
done

lemma *validNF-bind* [*wp-split*]:
$[\![\ \bigwedge x.\ \{\!|B\ x|\!\}\ g\ x\ \{\!|C|\!\}!;\ \{\!|A|\!\}\ f\ \{\!|B|\!\}!\ ]\!] \Longrightarrow$
$\{\!|A|\!\}\ do\ x \leftarrow f;\ g\ x\ od\ \{\!|C|\!\}!$
apply (*rule validNF*)
apply (*metis validNF-valid hoare-seq-ext*)
apply (*clarsimp simp*: *no-fail-def validNF-def bind-def' valid-def*)
apply *blast*
done

lemmas *validNF-seq-ext* = *validNF-bind*

24.3 validNF compound rules

lemma *validNF-state-assert* [*wp*]:
$\{\!|\ \lambda s.\ P\ ()\ s \wedge G\ s\ \ |\!\}\ state\text{-}assert\ G\ \{\!|\ P\ |\!\}!$
apply (*rule validNF*)
apply *wpsimp*
apply (*clarsimp simp*: *no-fail-def state-assert-def*
bind-def' assert-def return-def get-def)
done

lemma *validNF-modify* [*wp*]:
$\{\!|\ \lambda s.\ P\ ()\ (f\ s)\ |\!\}\ modify\ f\ \{\!|\ P\ |\!\}!$
apply (*clarsimp simp*: *modify-def*)

apply *wp*
done

lemma *validNF-gets* [*wp*]:
$\{\!|\lambda s.\ P\ (f\ s)\ s|\!\}$ *gets f* $\{\!|P|\!\}!$
apply (*clarsimp simp*: *gets-def*)
apply *wp*
done

lemma *validNF-condition* [*wp*]:
$[\![\ \{\!|\ Q\ |\!\}\ A\ \{\!|P|\!\}!;\ \{\!|\ R\ |\!\}\ B\ \{\!|P|\!\}!]\!] \Longrightarrow \{\!|\lambda s.\ \mathit{if}\ C\ s\ \mathit{then}\ Q\ s\ \mathit{else}\ R\ s|\!\}\ \mathit{condition}\ C\ A\ B\ \{\!|P|\!\}!$
apply *rule*
 apply (*drule validNF-valid*)+
 apply (*erule* (*1*) *condition-wp*)
apply (*drule validNF-no-fail*)+
apply (*clarsimp simp*: *no-fail-def condition-def*)
done

lemma *validNF-alt-def*:
$\mathit{validNF}\ P\ m\ Q = (\forall s.\ P\ s \longrightarrow ((\forall (r', s') \in \mathit{fst}\ (m\ s).\ Q\ r'\ s') \wedge \neg\ \mathit{snd}\ (m\ s)))$
by (*fastforce simp*: *validNF-def valid-def no-fail-def*)

lemma *validNF-assert* [*wp*]:
$\{\!|\ (\lambda s.\ P)\ \mathit{and}\ (R\ ())\ |\!\}\ \mathit{assert}\ P\ \{\!|\ R\ |\!\}!$
apply (*rule validNF*)
 apply (*clarsimp simp*: *valid-def in-return*)
apply (*clarsimp simp*: *no-fail-def return-def*)
done

lemma *validNF-false-pre*:
$\{\!|\ \lambda\text{-}.\ \mathit{False}\ |\!\}\ P\ \{\!|\ Q\ |\!\}!$
by (*clarsimp simp*: *validNF-def no-fail-def*)

lemma *validNF-chain*:
$[\![\{\!|P'|\!\}\ a\ \{\!|R'|\!\}!;\ \bigwedge s.\ P\ s \Longrightarrow P'\ s;\ \bigwedge r\ s.\ R'\ r\ s \Longrightarrow R\ r\ s]\!] \Longrightarrow \{\!|P|\!\}\ a\ \{\!|R|\!\}!$
by (*fastforce simp*: *validNF-def valid-def no-fail-def Ball-def*)

lemma *validNF-case-prod* [*wp*]:
$[\![\ \bigwedge x\ y.\ \mathit{validNF}\ (P\ x\ y)\ (B\ x\ y)\ Q\ ]\!] \Longrightarrow \mathit{validNF}\ (\mathit{case\text{-}prod}\ P\ v)\ (\mathit{case\text{-}prod}\ (\lambda x\ y.\ B\ x\ y)\ v)\ Q$
by (*metis prod.exhaust split-conv*)

lemma *validE-NF-case-prod* [*wp*]:
$[\![\ \bigwedge a\ b.\ \{\!|P\ a\ b|\!\}\ f\ a\ b\ \{\!|Q|\!\}, \{\!|E|\!\}!\ ]\!] \Longrightarrow$
$\{\!|\mathit{case}\ x\ \mathit{of}\ (a, b) \Rightarrow P\ a\ b|\!\}\ \mathit{case}\ x\ \mathit{of}\ (a, b) \Rightarrow f\ a\ b\ \{\!|Q|\!\}, \{\!|E|\!\}!$
apply (*clarsimp simp*: *validE-NF-alt-def*)
apply (*erule validNF-case-prod*)
done

lemma *no-fail-is-validNF-True*: *no-fail P s* = ($\{\!|$ *P* $|\!\}$ *s* $\{\!|$ λ- -. *True* $|\!\}$!)
by (*clarsimp simp*: *no-fail-def validNF-def valid-def*)

24.4 validNF reasoning in the exception monad

lemma *validE-NF* [*intro?*]:
$[\![$ $\{\!|$ *P* $|\!\}$ *f* $\{\!|$ *Q* $|\!\}$,$\{\!|$ *E* $|\!\}$; *no-fail P f* $]\!]$ $\Longrightarrow$ $\{\!|$ *P* $|\!\}$ *f* $\{\!|$ *Q* $|\!\}$,$\{\!|$ *E* $|\!\}$!
apply (*clarsimp simp*: *validE-NF-def*)
done

lemma *validE-NF-valid*:
$[\![$ $\{\!|$ *P* $|\!\}$ *f* $\{\!|$ *Q* $|\!\}$,$\{\!|$ *E* $|\!\}$! $]\!]$ $\Longrightarrow$ $\{\!|$ *P* $|\!\}$ *f* $\{\!|$ *Q* $|\!\}$,$\{\!|$ *E* $|\!\}$
apply (*clarsimp simp*: *validE-NF-def*)
done

lemma *validE-NF-no-fail*:
$[\![$ $\{\!|$ *P* $|\!\}$ *f* $\{\!|$ *Q* $|\!\}$,$\{\!|$ *E* $|\!\}$! $]\!]$ $\Longrightarrow$ *no-fail P f*
apply (*clarsimp simp*: *validE-NF-def*)
done

lemma *validE-NF-weaken-pre*[*wp-pre*]:
$[\![\{\!|Q|\!\}$ *a* $\{\!|R|\!\},\{\!|E|\!\}$!; $\bigwedge s.\ P\ s \Longrightarrow Q\ s]\!]$ $\Longrightarrow$ $\{\!|P|\!\}$ *a* $\{\!|R|\!\},\{\!|E|\!\}$!
apply (*clarsimp simp*: *validE-NF-alt-def*)
apply (*erule validNF-weaken-pre*)
apply *simp*
done

lemma *validE-NF-post-comb-conj-L*:
$[\![$ $\{\!|P|\!\}$ *f* $\{\!|Q|\!\}$, $\{\!|$ *E* $|\!\}$!; $\{\!|P'|\!\}$ *f* $\{\!|Q'|\!\}$, $\{\!|$ λ- -. *True* $|\!\}$ $]\!]$ $\Longrightarrow$ $\{\!|\lambda s.\ P\ s \wedge P'\ s$ $|\!\}$ *f* $\{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s|\!\}$, $\{\!|$ *E* $|\!\}$!
apply (*clarsimp simp*: *validE-NF-alt-def validE-def validNF-def*
valid-def no-fail-def split: *sum.splits*)
apply *force*
done

lemma *validE-NF-post-comb-conj-R*:
$[\![$ $\{\!|P|\!\}$ *f* $\{\!|Q|\!\}$, $\{\!|$ λ- -. *True* $|\!\}$; $\{\!|P'|\!\}$ *f* $\{\!|Q'|\!\}$, $\{\!|$ *E* $|\!\}$! $]\!]$ $\Longrightarrow$ $\{\!|\lambda s.\ P\ s \wedge P'\ s$ $|\!\}$ *f* $\{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s|\!\}$, $\{\!|$ *E* $|\!\}$!
apply (*clarsimp simp*: *validE-NF-alt-def validE-def validNF-def*
valid-def no-fail-def split: *sum.splits*)
apply *force*
done

lemma *validE-NF-post-comb-conj*:
$[\![$ $\{\!|P|\!\}$ *f* $\{\!|Q|\!\}$, $\{\!|$ *E* $|\!\}$!; $\{\!|P'|\!\}$ *f* $\{\!|Q'|\!\}$, $\{\!|$ *E* $|\!\}$! $]\!]$ $\Longrightarrow$ $\{\!|\lambda s.\ P\ s \wedge P'\ s$ $|\!\}$ *f* $\{\!|\lambda rv\ s.\ Q\ rv\ s \wedge Q'\ rv\ s|\!\}$, $\{\!|$ *E* $|\!\}$!
apply (*clarsimp simp*: *validE-NF-alt-def validE-def validNF-def*
valid-def no-fail-def split: *sum.splits*)

apply *force*
done

lemma *validE-NF-chain*:
$[\![\{\!|P'|\!\}\ a\ \{\!|R'|\!\},\{\!|E'|\!\}!;$
$\bigwedge s.\ P\ s \Longrightarrow P'\ s;$
$\bigwedge r'\ s'.\ R'\ r'\ s' \Longrightarrow R\ r'\ s';$
$\bigwedge r''\ s''.\ E'\ r''\ s'' \Longrightarrow E\ r''\ s'']\!] \Longrightarrow$
$\{\!|\lambda s.\ P\ s\ |\!\}\ a\ \{\!|\lambda r'\ s'.\ R\ r'\ s'|\!\},\{\!|\lambda r''\ s''.\ E\ r''\ s''|\!\}!$
by (*fastforce simp*: *validE-NF-def validE-def2 no-fail-def Ball-def split*: *sum.splits*)

lemma *validE-NF-bind-wp* [*wp*]:
$[\![\bigwedge x.\ \{\!|B\ x|\!\}\ g\ x\ \{\!|C|\!\},\ \{\!|E|\!\}!;\ \{\!|A|\!\}\ f\ \{\!|B|\!\},\ \{\!|E|\!\}!]\!] \Longrightarrow \{\!|A|\!\}\ f >>=E\ (\lambda x.\ g\ x)\ \{\!|C|\!\},$
$\{\!|E|\!\}!$
apply (*unfold validE-NF-alt-def bindE-def*)
apply (*rule validNF-bind* [*rotated*])
apply *assumption*
apply (*clarsimp simp*: *lift-def throwError-def split*: *sum.splits*)
apply *wpsimp*
done

lemma *validNF-catch* [*wp*]:
$[\![\bigwedge x.\ \{\!|E\ x|\!\}\ handler\ x\ \{\!|Q|\!\}!;\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},\ \{\!|E|\!\}!]\!] \Longrightarrow \{\!|P|\!\}\ f\ <catch>\ (\lambda x.\ handler$
$x)\ \{\!|Q|\!\}!$
apply (*unfold validE-NF-alt-def catch-def*)
apply (*rule validNF-bind* [*rotated*])
apply *assumption*
apply (*clarsimp simp*: *lift-def throwError-def split*: *sum.splits*)
apply *wp*
done

lemma *validNF-throwError* [*wp*]:
$\{\!|E\ e|\!\}\ throwError\ e\ \{\!|P|\!\},\ \{\!|E|\!\}!$
by (*unfold validE-NF-alt-def throwError-def o-def*) *wpsimp*

lemma *validNF-returnOk* [*wp*]:
$\{\!|P\ e|\!\}\ returnOk\ e\ \{\!|P|\!\},\ \{\!|E|\!\}!$
by (*clarsimp simp*: *validE-NF-alt-def returnOk-def*) *wpsimp*

lemma *validNF-whenE* [*wp*]:
$(P \Longrightarrow \{\!|Q|\!\}\ f\ \{\!|R|\!\},\ \{\!|E|\!\}!) \Longrightarrow \{\!|if\ P\ then\ Q\ else\ R\ ()|\!\}\ whenE\ P\ f\ \{\!|R|\!\},\ \{\!|E|\!\}!$
unfolding *whenE-def* **by** *clarsimp wp*

lemma *validNF-nobindE* [*wp*]:
$[\![\ \{\!|B|\!\}\ g\ \{\!|C|\!\},\{\!|E|\!\}!;$
$\{\!|A|\!\}\ f\ \{\!|\lambda r\ s.\ B\ s|\!\},\{\!|E|\!\}!\ ]\!] \Longrightarrow$
$\{\!|A|\!\}\ doE\ f;\ g\ odE\ \{\!|C|\!\},\{\!|E|\!\}!$
by *clarsimp wp*

Setup triple rules for *validE-NF* so that we can use wp combinator rules.

definition *validE-NF-property* $Q\ E\ s\ b \equiv \neg\ snd\ (b\ s)$
$\wedge\ (\forall\,(r',\ s') \in fst\ (b\ s).\ case\ r'\ of\ Inl\ x \Rightarrow E\ x\ s' \mid Inr\ x \Rightarrow Q\ x\ s')$

lemma *validE-NF-is-triple* [*wp-trip*]:
validE-NF P f Q E = *triple-judgement P f* (*validE-NF-property Q E*)
apply (*clarsimp simp*: *validE-NF-def validE-def2 no-fail-def triple-judgement-def validE-NF-property-def split*: *sum.splits*)
apply *blast*
done

lemma *validNF-cong*:
$[\![\ \bigwedge s.\ P\ s = P'\ s;\ \bigwedge s.\ P\ s \Longrightarrow m\ s = m'\ s;$
$\bigwedge r'\ s'\ s.\ [\![\ P\ s;\ (r',\ s') \in fst\ (m\ s)\ ]\!] \Longrightarrow Q\ r'\ s' = Q'\ r'\ s'\ ]\!] \Longrightarrow$
$(\{\!|\ P\ |\!\}\ m\ \{\!|\ Q\ |\!\}!) = (\{\!|\ P'\ |\!\}\ m'\ \{\!|\ Q'\ |\!\}!)$
by (*fastforce simp*: *validNF-alt-def*)

lemma *validE-NF-liftE* [*wp*]:
$\{\!|P|\!\}\ f\ \{\!|Q|\!\}! \Longrightarrow \{\!|P|\!\}\ liftE\ f\ \{\!|Q|\!\},\{\!|E|\!\}!$
by (*wpsimp simp*: *validE-NF-alt-def liftE-def*)

lemma *validE-NF-handleE'* [*wp*]:
$[\![\ \bigwedge x.\ \{\!|F\ x|\!\}\ handler\ x\ \{\!|Q|\!\},\{\!|E|\!\}!;\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|F|\!\}!\ ]\!] \Longrightarrow$
$\{\!|P|\!\}\ f$ <*handle2*> $(\lambda x.\ handler\ x)\ \{\!|Q|\!\},\{\!|E|\!\}!$
apply (*unfold validE-NF-alt-def handleE'-def*)
apply (*rule validNF-bind* [*rotated*])
apply *assumption*
apply (*clarsimp split*: *sum.splits*)
apply *wpsimp*
done

lemma *validE-NF-handleE* [*wp*]:
$[\![\ \bigwedge x.\ \{\!|F\ x|\!\}\ handler\ x\ \{\!|Q|\!\},\{\!|E|\!\}!;\ \{\!|P|\!\}\ f\ \{\!|Q|\!\},\{\!|F|\!\}!\ ]\!] \Longrightarrow$
$\{\!|P|\!\}\ f$ <*handle*> $handler\ \{\!|Q|\!\},\{\!|E|\!\}!$
apply (*unfold handleE-def*)
apply (*metis validE-NF-handleE'*)
done

lemma *validE-NF-condition* [*wp*]:
$[\![\ \{\!|\ Q\ |\!\}\ A\ \{\!|P|\!\},\{\!|\ E\ |\!\}!;\ \{\!|\ R\ |\!\}\ B\ \{\!|P|\!\},\{\!|\ E\ |\!\}!]\!]$
$\Longrightarrow \{\!|\lambda s.\ if\ C\ s\ then\ Q\ s\ else\ R\ s|\!\}\ condition\ C\ A\ B\ \{\!|P|\!\},\{\!|\ E\ |\!\}!$
apply *rule*
apply (*drule validE-NF-valid*)+
apply *wp*
apply (*drule validE-NF-no-fail*)+
apply (*clarsimp simp*: *no-fail-def condition-def*)
done

Strengthen setup.

context *strengthen-implementation* **begin**

lemma *strengthen-hoare* [*strg*]:
$(\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (Q\ r\ s)\ (R\ r\ s))$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\{\!|P|\!\}\ f\ \{\!|Q|\!\})\ (\{\!|P|\!\}\ f\ \{\!|R|\!\})$
by (*cases F*, *auto elim*: *hoare-strengthen-post*)

lemma *strengthen-validE-R-cong*[*strg*]:
$(\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (Q\ r\ s)\ (R\ r\ s))$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\{\!|P|\!\}\ f\ \{\!|Q|\!\},\ -)\ (\{\!|P|\!\}\ f\ \{\!|R|\!\},\ -)$
by (*cases F*, *auto intro*: *hoare-post-imp-R*)

lemma *strengthen-validE-cong*[*strg*]:
$(\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (Q\ r\ s)\ (R\ r\ s))$
$\Longrightarrow (\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (S\ r\ s)\ (T\ r\ s))$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\{\!|P|\!\}\ f\ \{\!|Q|\!\},\ \{\!|S|\!\})\ (\{\!|P|\!\}\ f\ \{\!|R|\!\},\ \{\!|T|\!\})$
by (*cases F*, *auto elim*: *hoare-post-impErr*)

lemma *strengthen-validE-E-cong*[*strg*]:
$(\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (S\ r\ s)\ (T\ r\ s))$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\{\!|P|\!\}\ f\ -,\ \{\!|S|\!\})\ (\{\!|P|\!\}\ f\ -,\ \{\!|T|\!\})$
by (*cases F*, *auto elim*: *hoare-post-impErr simp*: *validE-E-def*)

lemma *wpfix-strengthen-hoare*:
$(\bigwedge s.\ st\ (\neg\ F)\ (\longrightarrow)\ (P\ s)\ (P'\ s))$
$\Longrightarrow (\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (Q\ r\ s)\ (Q'\ r\ s))$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\{\!|P|\!\}\ f\ \{\!|Q|\!\})\ (\{\!|P'|\!\}\ f\ \{\!|Q'|\!\})$
by (*cases F*, *auto elim*: *hoare-chain*)

lemma *wpfix-strengthen-validE-R-cong*:
$(\bigwedge s.\ st\ (\neg\ F)\ (\longrightarrow)\ (P\ s)\ (P'\ s))$
$\Longrightarrow (\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (Q\ r\ s)\ (Q'\ r\ s))$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\{\!|P|\!\}\ f\ \{\!|Q|\!\},\ -)\ (\{\!|P'|\!\}\ f\ \{\!|Q'|\!\},\ -)$
by (*cases F*, *auto elim*: *hoare-chainE simp*: *validE-R-def*)

lemma *wpfix-strengthen-validE-cong*:
$(\bigwedge s.\ st\ (\neg\ F)\ (\longrightarrow)\ (P\ s)\ (P'\ s))$
$\Longrightarrow (\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (Q\ r\ s)\ (R\ r\ s))$
$\Longrightarrow (\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (S\ r\ s)\ (T\ r\ s))$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\{\!|P|\!\}\ f\ \{\!|Q|\!\},\ \{\!|S|\!\})\ (\{\!|P'|\!\}\ f\ \{\!|R|\!\},\ \{\!|T|\!\})$
by (*cases F*, *auto elim*: *hoare-chainE*)

lemma *wpfix-strengthen-validE-E-cong*:
$(\bigwedge s.\ st\ (\neg\ F)\ (\longrightarrow)\ (P\ s)\ (P'\ s))$
$\Longrightarrow (\bigwedge r\ s.\ st\ F\ (\longrightarrow)\ (S\ r\ s)\ (T\ r\ s))$
$\Longrightarrow st\ F\ (\longrightarrow)\ (\{\!|P|\!\}\ f\ -,\ \{\!|S|\!\})\ (\{\!|P'|\!\}\ f\ -,\ \{\!|T|\!\})$
by (*cases F*, *auto elim*: *hoare-chainE simp*: *validE-E-def*)

lemma *wpfix-no-fail-cong*:
$(\bigwedge s.\ st\ (\neg\ F)\ (\longrightarrow)\ (P\ s)\ (P'\ s))$

$\Longrightarrow$ *st F* $(\longrightarrow)$ (*no-fail P f*) (*no-fail P′ f*)
by (*cases F*, *auto elim*: *no-fail-pre*)

lemmas *nondet-wpfix-strgs* =
wpfix-strengthen-validE-R-cong
wpfix-strengthen-validE-E-cong
wpfix-strengthen-validE-cong
wpfix-strengthen-hoare
wpfix-no-fail-cong

end

lemmas *nondet-wpfix-strgs*[*wp-fix-strgs*]
= *strengthen-implementation.nondet-wpfix-strgs*

end

25 lsm hooks [lsm$_h$*ooks*]

theory *LSM-Cap*
imports
Element
../lib/Monad-WP/NonDetMonadVCG

begin
definition *ns-capable* :: *user-namespace* $\Rightarrow$ *int* $\Rightarrow$ *bool*
where *ns-capable ns cap* $\equiv$ *True*

definition *cap-inode-setxattr* :: *′s* $\Rightarrow$ *dentry* $\Rightarrow$*xattr* $\Rightarrow$*string*$\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ (*′s*, *int*) *nondet-monad*
where *cap-inode-setxattr s dentry name value size′ flags′* $\equiv$ *do*
ns $\leftarrow$ *return* (*s-user-ns* (*d-sb dentry*));
rc $\leftarrow$ (*if name* $\neq$ *XATTR-SECURITY-PREFIX then return 0 else*
if name = *XATTR-NAME-CAPS then return 0 else*
if $\neg$(*ns-capable ns CAP-SYS-ADMIN*) *then return* (−*EPERM*) *else*
return 0);
return(*rc*)
od

definition *CAP-TO-INDEX x* $\equiv$ ((*x*) >> *5*)

definition *CAP-TO-MASK x* $\equiv$ (*1* << (*nat*(*x AND 31*)))
term (*kcap k*) ! (*nat*(*CAP-TO-INDEX flag*))

definition *cap-raised* :: *kernel-cap-t* $\Rightarrow$ *int* $\Rightarrow$ *int*
where *cap-raised k flag* $\equiv$ ((*kcap k*) ! (*nat*(*CAP-TO-INDEX flag*))) *AND* (*CAP-TO-MASK flag*)

end

26 lsm hooks [lsm$_{h}ooks$]

theory *Linux-LSM-Hooks*
imports
Element
../lib/Monad-WP/NonDetMonadVCG
SOAC

begin

In this theory, we introduce LSM hooks

26.1 lsm hook

locale *lsm-superblock-hooks* =
fixes *s0* :: *'s*
fixes *state* :: *'s*
fixes *sb-security* ::*'s* ⇒*super-block* ⇒*'sbsec option*
fixes *hook-sb-alloc* :: *'s* ⇒ *super-block* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-free* :: *'s* ⇒ *super-block* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-sb-copy-data* :: *'s* ⇒ *string* ⇒ *string* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-remount* :: *'s* ⇒ *super-block* ⇒ *Void* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-kern-mount* :: *'s* ⇒ *super-block* ⇒ *int* ⇒ *string* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-show-options* :: *'s* ⇒ *seq-file* ⇒ *super-block* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-statfs* :: *'s* ⇒ *dentry* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-mount* :: *'s* ⇒ *string* ⇒ *path* ⇒ *string* ⇒ *int* ⇒ *Void* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-sb-umount* :: *'s* ⇒ *vfsmount* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-pivotroot* :: *'s* ⇒ *path* ⇒ *path* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-set-mnt-opts* :: *'s* ⇒ *super-block* ⇒ *opts* ⇒ *nat* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-clone-mnt-opts* :: *'s* ⇒ *super-block* ⇒ *super-block* ⇒ *int* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sb-parse-opts-str* :: *'s*⇒*string* ⇒ *opts* ⇒ (*'s*, *int*) *nondet-monad*

assumes *stb-sb-alloc-hook* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}sb\text{-}alloc\ s\ sb\ \ \{\!|\lambda r\ s.\ r = 0 \vee r = -ENOMEM|\!\}$
assumes *stb-sb-free* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}sb\text{-}free\ s\ sb\ \{\!|\lambda r\ s.\ r = unit\ |\!\}$
assumes *stb-sb-copy-data* :
$\{\!|\lambda s.\ True|\!\}\ hook\text{-}sb\text{-}copy\text{-}data\ s\ orig\ smackopts\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r = (uminus\ 12))|\!\}$
assumes *stb-sb-remount* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}sb\text{-}remount\ s\ sb\ data'\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-sb-kern-mount* :

$\bigwedge sa.$ $\{\lambda s\ .\ s = sa\ \}$ *hook-sb-kern-mount s sb flag data* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-sb-show-options* :
$\bigwedge sa.$ $\{\lambda s\ .\ s = sa\ \}$ *hook-sb-show-options s sq sb* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-sb-statfs* :
$\bigwedge sa.$ $\{\lambda s\ .\ s = sa\ \}$ *hook-sb-statfs s d* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-sb-mount* :
$\bigwedge sa.$ $\{\lambda s\ .\ s = sa\ \}$ *hook-sb-mount s devname path type flag data'*
$\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-sb-umount*:
$\bigwedge sa.$ $\{\lambda s\ .\ s = sa\ \}$ *hook-sb-umount s mnt' flag* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-sb-pivotroot*:
$\bigwedge sa.$ $\{\lambda s\ .\ s = sa\ \}$ *hook-sb-pivotroot s old-path new-path*
$\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-sb-set-mnt-opts* :
$\bigwedge sa.$ $\{\lambda s\ .\ s = sa\ \}$ *hook-sb-set-mnt-opts s sb opt kflag sflag*
$\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-sb-clone-mnt-opts* :
$\bigwedge sa.$ $\{\lambda s\ .\ s = sa\ \}$ *hook-sb-clone-mnt-opts s oldsb newsb kflag sflag*
$\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-parse-opts-str* :
$\bigwedge sa.$ $\{\lambda s\ .\ s = sa\ \}$ *hook-sb-parse-opts-str s str opt* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$

locale *lsm-task-hooks* =
fixes *s0* :: *'s*

fixes *t-security* :: *'s* ⇒ *Cred* ⇒ *'tsec option*
fixes *hook-task-alloc* :: *'s* ⇒ *Task* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-free* :: *'s* ⇒ *Task* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-cred-alloc-blank* :: *'s* ⇒ *Cred* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-cred-free* :: *'s* ⇒ *Cred* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-prepare-creds* :: *'s* ⇒ *Cred* ⇒ *Cred* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-transfer-creds* :: *'s* ⇒ *Cred* ⇒ *Cred* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-cred-getsecid* :: *'s* ⇒ *Cred* ⇒ *u32* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-task-fix-setuid* :: *'s* ⇒ *Cred* ⇒ *Cred* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-setpgid* :: *'s* ⇒ *Task* ⇒ *pid-t* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-getpgid* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-getsid* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-getsecid* :: *'s* ⇒ *Task* ⇒ *u32* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-task-setnice* :: *'s* ⇒ *Task* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-setioprio* :: *'s* ⇒ *Task* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-getioprio* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-prlimit* :: *'s* ⇒ *Cred* ⇒ *Cred* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-setrlimit* :: *'s* ⇒ *Task* ⇒ *nat* ⇒ *rlimit* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-setscheduler* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-task-getscheduler* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*

fixes *hook-task-movememory* :: $'s \Rightarrow Task \Rightarrow ('s, int)$ *nondet-monad*
fixes *hook-task-kill* :: $'s \Rightarrow Task \Rightarrow siginfo \Rightarrow int \Rightarrow Cred\ option \Rightarrow ('s, int)$ *nondet-monad*
fixes *hook-task-prctl* :: $'s \Rightarrow int \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow ('s, int)$ *nondet-monad*
fixes *hook-task-to-inode* :: $'s \Rightarrow Task \Rightarrow inode \Rightarrow ('s, unit)$ *nondet-monad*

assumes *stb-task-alloc* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}alloc\ s\ task\ cflag\ \{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-task-free* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}free\ s\ task\ \{\!|\lambda r\ s.\ r = unit\ |\!\}$
assumes *stb-cred-alloc-blank* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}cred\text{-}alloc\text{-}blank\ s\ cred'\ gfp'\ \{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-cred-free* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}cred\text{-}free\ s\ c\ \{\!|\lambda r\ s.\ r = unit\ |\!\}$
assumes *stb-prepare-creds* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}prepare\text{-}creds\ s\ new'\ old\ gfp'\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-transfer-creds* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}transfer\text{-}creds\ s\ new'\ old\ \{\!|\lambda r\ s.\ r = unit|\!\}$
assumes *stb-task-setpgid*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}setpgid\ s\ t\ pid\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-task-getpgid*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}getpgid\ s\ t\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-task-getsid*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}getsid\ s\ t\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-task-getsecid*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}getsecid\ s\ t\ secid'\ \{\!|\lambda r\ s.\ r = unit|\!\}$
assumes *stb-task-setnice*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}setnice\ s\ t\ nice\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-task-setioprio*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}setioprio\ s\ t\ ioprio\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-task-getioprio*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}getioprio\ s\ t\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-task-setrlimit*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}setrlimit\ s\ t\ resource\ new\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-task-setscheduler*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}setscheduler\ s\ t\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-task-getscheduler*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}getscheduler\ s\ t\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-task-movememory*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}task\text{-}movememory\ s\ t\ \{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r$

$\neq 0)\}$
assumes *stb-task-kill*:
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}\ \textit{hook-task-kill s t info sig c}'\ \{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-task-prctl* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$
$\textit{hook-task-prctl s opt}'\ \textit{arg2 arg3 arg4 arg5}$
$\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-task-to-inode* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}\ \textit{hook-task-to-inode s t inode}\ \{\lambda r\ s.\ r = \textit{unit}\ \}$

locale *lsm-binder-hooks* =
fixes *s0* :: *'s*
fixes *hook-binder-set-context-mgr* :: *'s* ⇒ *Task* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-binder-transaction* :: *'s* ⇒ *Task* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-binder-transfer-binder* :: *'s* ⇒ *Task* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-binder-transfer-file* :: *'s* ⇒ *Task* ⇒ *Task* ⇒*Files* ⇒ (*'s*, *int*) *nondet-monad*

assumes *stb-binder-set-context-mgr* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}\ \textit{hook-binder-set-context-mgr s mgr}\ \ \{\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-binder-transaction* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}\ \textit{hook-binder-transaction s from to}\ \ \{\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-binder-transfer-binder* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}\ \textit{hook-binder-transfer-binder s from to}\ \ \{\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-binder-transfer-file*:
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}\ \textit{hook-binder-transfer-file s from to file}\ \ \{\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)\}$

locale *lsm-ptrace-hooks* =
fixes *s0* :: *'s*

fixes *hook-ptrace-access-check* :: *'s* ⇒ *Task* ⇒ *nat* ⇒(*'s*, *int*) *nondet-monad*

fixes *hook-ptrace-traceme* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
assumes *stb-ptrace-access-check* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}\ \textit{hook-ptrace-access-check s child m}\ \{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *stb-ptrace-traceme*:
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}\ \textit{hook-ptrace-traceme s parent}'\ \{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$

locale *lsm-capable-hooks* =
fixes *s0* :: *'s*

fixes *hook-capget* :: $'s \Rightarrow$ *Task* $\Rightarrow$ *kct* $\Rightarrow$ *kct* $\Rightarrow$*kct* $\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-capset* :: $'s \Rightarrow$ *Cred* $\Rightarrow$ *Cred* $\Rightarrow$ *kct* $\Rightarrow$ *kct* $\Rightarrow$*kct* $\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-capable* :: $'s \Rightarrow$ *Cred* $\Rightarrow$ *ns* $\Rightarrow$ *cap* $\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-capable-noaudit* :: $'s \Rightarrow$ *Cred* $\Rightarrow$ *ns* $\Rightarrow$ *cap* $\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-quotactl*:: $'s \Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *super-block option*$\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-quota-on*:: $'s \Rightarrow$ *dentry*$\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-syslog* :: $'s \Rightarrow$ *int* $\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-settime64* :: $'s \Rightarrow$ *ts* $\Rightarrow$ *tz option* $\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-vm-enough-memory-mm* :: $'s \Rightarrow$ *mm* $\Rightarrow$ *pages* $\Rightarrow$ $('s,$ *int*) *nondet-monad*
assumes *stb-capget* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-capget s target effective inheritable permitted*
$\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-capset* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-capset s new old effective inheritable permitted*
$\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-capable* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-capable s c ns cap* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-capable-noaudit* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-capable-noaudit s c ns cap* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-quotactl* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-quotactl s cmds t id' sb* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-quota-on* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-quota-on s dentry* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-syslog* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-syslog s type* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-settime64* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-settime64 s ts tz* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *stb-vm-enough-memory-mm* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-vm-enough-memory-mm s mm' pages* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

locale *lsm-bprm-hooks* =
fixes *s0* :: $'s$
fixes *hook-bprm-set-creds* :: $'s \Rightarrow$*linux-binprm* $\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-bprm-check* :: $'s \Rightarrow$*linux-binprm* $\Rightarrow$ $('s,$ *int*) *nondet-monad*
fixes *hook-bprm-committing-creds* :: $'s \Rightarrow$*linux-binprm* $\Rightarrow$ $('s,$ *unit*) *nondet-monad*
fixes *hook-bprm-committed-creds* :: $'s \Rightarrow$*linux-binprm* $\Rightarrow$ $('s,$ *unit*) *nondet-monad*

assumes *stb-bprm-set-creds* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-bprm-set-creds s bprm* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *stb-bprm-check* :
$\bigwedge sa.$ {|λs . $s = sa$ |} *hook-bprm-check s bprm* {|$\lambda r\ s.$ $s = sa \wedge (r = 0 \vee r \neq 0)$|}
assumes *stb-bprm-committing-creds* :
$\bigwedge sa.$ {|λs . $s = sa$ |} *hook-bprm-committing-creds s bprm* {|$\lambda r\ s.$ $r = unit$|}
assumes *stb-bprm-committed-creds* :
$\bigwedge sa.$ {|λs . $s = sa$ |} *hook-bprm-committed-creds s bprm* {|$\lambda r\ s.$ $r = unit$|}

locale *lsm-file-hooks* =
fixes *s0* :: *'s*
fixes *access* :: *'s* ⇒ *Subj* ⇒ *Obj* ⇒ *access* ⇒ *bool*
fixes *current* :: *'s* ⇒ *process-id*
fixes *f-security* :: *'s* ⇒ *Files* ⇒ *'fsec option*
fixes *hook-file-permission* :: *'s* ⇒ *Files* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-file-alloc* :: *'s* ⇒ *Files* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-file-free* :: *'s* ⇒ *Files* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-file-ioctl* :: *'s* ⇒ *Files* ⇒ *IOC-DIR* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-mmap-file* :: *'s* ⇒ *Files option* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-mmap-addr* :: *'s* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-file-mprotect* :: *'s* ⇒ *vm-area-struct* ⇒ *nat* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-file-lock* :: *'s* ⇒ *Files* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-file-fcntl* :: *'s* ⇒ *Files* ⇒ *nat* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-file-set-fowner* :: *'s* ⇒ *Files* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-file-send-sigiotask* :: *'s* ⇒ *Task* ⇒ *fown-struct* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-file-receive* :: *'s* ⇒ *Files* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-file-open* :: *'s* ⇒ *Files* ⇒ (*'s*, *int*) *nondet-monad*

assumes *stb-file-permission* :
$\bigwedge$*sa file mask'*. {|λs . $s = sa$ |}
hook-file-permission sa file mask'
{|$\lambda r\ s.$ $s = sa \wedge (r = 0 \vee r \neq 0)$ |}
assumes *stb-file-alloc-security* :
$\bigwedge$*sa file*. {|λs . $s = sa \wedge$ *f-security s file* = *None* |}
hook-file-alloc sa file
{|$\lambda r\ s.$ ($r = 0 \wedge$ *f-security s file* ≠ *None* ∧ $s \neq sa$) ∨ ($r \neq 0 \wedge s = sa$) |}
assumes *stb-file-free-security* :
$\bigwedge$*sa file*. {|λs . $s = sa$|} *hook-file-free sa file* {|$\lambda r\ s.$ $r = unit \wedge$ *f-security s file* = *None* |}
assumes *stb-file-ioctl* :
$\bigwedge sa.$ {|$\lambda s.$ $s = sa$|} *hook-file-ioctl sa file cmd arg* {|$\lambda r\ s.$ $s = sa$ |}
assumes *file-ioctl-ac* :
(∃ *p*. *access s* (*current s*) (*File file*) *p* = *True* ⟶
{|$\lambda s.$ *True*|} *hook-file-ioctl sa file cmd arg* {|$\lambda r\ s.$ $r = 0$ |}) ∨
(∃ *p*. *access s* (*current s*) (*File file*) *p* = *False* ⟶
{|$\lambda s.$ *True*|} *hook-file-ioctl sa file cmd arg* {|$\lambda r\ s.$ $r \neq 0$ |})
assumes *stb-mmap-addr* :

$\bigwedge sa.\ \{\!|\lambda s.\ s = sa\ |\!\}\ hook\text{-}mmap\text{-}addr\ sa\ addr \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *stb-mmap-file* :
$\bigwedge sa.\ \{\!|\lambda s.\ s = sa\ |\!\}\ hook\text{-}mmap\text{-}file\ sa\ file'\ prot\ mprot\ flgs\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-file-mprotect* :
$\bigwedge sa.\ \{\!|\lambda s.\ s = sa\ |\!\}\ hook\text{-}file\text{-}mprotect\ sa\ vma\ reqprot\ prot\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-file-lock* :
$\bigwedge sa.\ \{\!|\lambda s.\ s = sa\ |\!\}\ hook\text{-}file\text{-}lock\ sa\ file\ fcmd\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-file-fcntl* :
$\bigwedge sa.\ \{\!|\lambda s.\ s = sa\ |\!\}\ hook\text{-}file\text{-}fcntl\ sa\ file\ fcmd\ arg\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-file-set-fowner* :
$\bigwedge sa\ file.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}file\text{-}set\text{-}fowner\ sa\ file\ \{\!|\lambda r\ s.\ r = unit|\!\}$
assumes *stb-file-send-sigiotask* :
$\bigwedge sa.\ \{\!|\lambda s.\ s = sa\ |\!\}\ hook\text{-}file\text{-}send\text{-}sigiotask\ sa\ tsk'\ fown\ sig\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-file-receive* :
$\bigwedge sa.\ \{\!|\lambda s.\ s = sa\ |\!\}\ hook\text{-}file\text{-}receive\ sa\ file\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-file-open* :
$\bigwedge sa.\ \{\!|\lambda s.\ s = sa\ |\!\}\ hook\text{-}file\text{-}open\ sa\ file\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

begin
end

locale *lsm-dentry-hooks* =

fixes $s0 :: 's$
fixes $hook\text{-}dentry\text{-}init\text{-}security :: 's \Rightarrow dentry \Rightarrow mode \Rightarrow string \Rightarrow string \Rightarrow int \Rightarrow ('s, int)\ nondet\text{-}monad$
fixes $hook\text{-}dentry\text{-}create\text{-}files\text{-}as :: 's \Rightarrow dentry \Rightarrow mode \Rightarrow string \Rightarrow Cred \Rightarrow Cred \Rightarrow ('s, int)\ nondet\text{-}monad$
assumes *stb-dentry-init-security* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$
$hook\text{-}dentry\text{-}init\text{-}security\ s\ dentry\ m\ name\ ctx\ xtxlen$
$\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-dentry-create-files-as* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$
$hook\text{-}dentry\text{-}create\text{-}files\text{-}as\ s\ dentry\ m\ name\ old\ new$
$\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

locale *lsm-inode-hooks* =
fixes $s0 :: 's$
fixes $i\text{-}security :: 's \Rightarrow inode \Rightarrow 'isec\ option$
fixes $hook\text{-}inode\text{-}alloc :: 's \Rightarrow inode \Rightarrow ('s, int)\ nondet\text{-}monad$

fixes *hook-inode-free* :: $'s \Rightarrow inode \Rightarrow ('s,\ unit)\ nondet\text{-}monad$
fixes *hook-inode-init-security* :: $'s \Rightarrow inode \Rightarrow inode \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow int \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-old-inode-init-security* :: $'s \Rightarrow inode \Rightarrow inode \Rightarrow qstr \Rightarrow string \Rightarrow string \Rightarrow int \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-create* :: $'s \Rightarrow inode \Rightarrow dentry \Rightarrow mode \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-link* :: $'s \Rightarrow dentry \Rightarrow inode \Rightarrow dentry \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-unlink* :: $'s \Rightarrow inode \Rightarrow dentry \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-symlink* :: $'s \Rightarrow inode \Rightarrow dentry \Rightarrow string \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-mkdir* :: $'s \Rightarrow inode \Rightarrow dentry \Rightarrow mode \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-rmdir* :: $'s \Rightarrow inode \Rightarrow dentry \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-mknod* :: $'s \Rightarrow inode \Rightarrow dentry \Rightarrow mode \Rightarrow dev\text{-}t => ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-rename* :: $'s \Rightarrow inode \Rightarrow dentry \Rightarrow inode \Rightarrow dentry \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-readlink* :: $'s \Rightarrow dentry \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-follow-link* :: $'s \Rightarrow dentry \Rightarrow inode \Rightarrow bool \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-permission* :: $'s \Rightarrow inode \Rightarrow mask \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-setattr* :: $'s \Rightarrow dentry \Rightarrow iattr \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-getattr* :: $'s \Rightarrow path \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-setxattr* :: $'s \Rightarrow dentry \Rightarrow xattr \Rightarrow string \Rightarrow int \Rightarrow flags \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-post-setxattr* :: $'s \Rightarrow dentry \Rightarrow xattr \Rightarrow string \Rightarrow int \Rightarrow flags \Rightarrow ('s,\ unit)\ nondet\text{-}monad$
fixes *hook-inode-getxattr* :: $'s \Rightarrow dentry \Rightarrow xattr \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-listxattr* :: $'s \Rightarrow dentry \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-removexattr* :: $'s \Rightarrow dentry \Rightarrow xattr \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-need-killpriv* :: $'s \Rightarrow dentry \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-killpriv* :: $'s \Rightarrow dentry \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-getsecurity* :: $'s \Rightarrow inode \Rightarrow xattr \Rightarrow Void \Rightarrow bool \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-setsecurity* :: $'s \Rightarrow inode \Rightarrow xattr \Rightarrow Void \Rightarrow nat \Rightarrow int \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-listsecurity* :: $'s \Rightarrow inode \Rightarrow Void \Rightarrow int \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-getsecid* :: $'s \Rightarrow inode \Rightarrow u32 \Rightarrow ('s,\ unit)\ nondet\text{-}monad$
fixes *hook-inode-copy-up* :: $'s \Rightarrow dentry \Rightarrow Cred\ option \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-copy-up-xattr* :: $'s \Rightarrow xattr \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-invalidate-secctx* :: $'s \Rightarrow inode \Rightarrow ('s,\ unit)\ nondet\text{-}monad$
fixes *hook-inode-notifysecctx* :: $'s \Rightarrow inode \Rightarrow string \Rightarrow u32 \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-setsecctx* :: $'s \Rightarrow dentry \Rightarrow string \Rightarrow u32 \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-inode-getsecctx* :: $'s \Rightarrow inode \Rightarrow string \Rightarrow u32 \Rightarrow ('s,\ int)\ nondet\text{-}monad$
assumes *stb-inode-alloc* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}alloc\ s\ inode\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-free* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}free\ s\ inode\ \{\!|\lambda r\ s.\ \ r = unit|\!\}$
assumes *stb-inode-init-security* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}init\text{-}security\ s\ inode\ dir\ qstr\ \ name\ value\ len'$
$\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *stb-old-inode-init-security* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}old\text{-}inode\text{-}init\text{-}security\ s\ inode\ dir\ qstr\ name\ value\ len'$
$\{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-create* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}create\ s\ dir\ dentry\ m\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-link* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}link\ s\ old\text{-}dentry\ dir\ new\text{-}dentry\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-unlink* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}unlink\ s\ dir\ dentry\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-symlink* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}symlink\ s\ dir\ dentry\ old\text{-}name\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-mkdir* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}mkdir\ s\ dir\ dentry\ m\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-rmdir* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}rmdir\ s\ dir\ dentry\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-mknod* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}mknod\ s\ dir\ dentry\ m\ dev\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-rename* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}rename\ s\ new\text{-}dir\ new\text{-}dentry\ old\text{-}dir\ old\text{-}dentry$

$\{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-readlink* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}readlink\ s\ \ dentry\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-follow-link* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}follow\text{-}link\ s\ dentry\ inode\ rcu'$
$\{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-permission* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}permission\ s\ inode\ m\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-setattr* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}setattr\ s\ dentry\ attr'\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-getattr* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}getattr\ s\ path\ \{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-setxattr* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}setxattr\ s\ dentry\ name'\ value\ size'\ flgs$
$\{\!|\lambda r\ s.\ \ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-inode-post-setxattr* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}post\text{-}setxattr\ \ s\ dentry\ name'\ value\ size'\ flgs$

$\{\!|\lambda r\ s.\ \ r = unit|\!\}$
assumes *stb-inode-getxattr* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}getxattr\ s\ dentry\ name'\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-listxattr* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}listxattr\ s\ dentry\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-removexattr* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}removexattr\ s\ dentry\ name'\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-need-killpriv* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}need\text{-}killpriv\ s\ dentry\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-killpriv* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}killpriv\ s\ dentry\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-getsecurity* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}getsecurity\ s\ inode\ name'\ buffer\ alloc\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-setsecurity* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}setsecurity\ s\ inode\ name'\ va\ size'\ flgs\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-listsecurity* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}listsecurity\ s\ inode\ buffer\ bsize\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-getsecid* : $\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}getsecid\ \ s\ inode\ secid'\ \{\!|\lambda r\ s.\ r = unit|\!\}$
assumes *stb-inode-copy-up* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}copy\text{-}up\ \ s\ src\ new\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-copy-up-xattr* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}copy\text{-}up\text{-}xattr\ \ s\ name'\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-invalidate-secctx* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}invalidate\text{-}secctx\ s\ inode\ \{\!|\lambda r\ s.\ \ r = unit|\!\}$
assumes *stb-inode-notifysecctx* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}notifysecctx\ s\ inode\ ctx\ ctxlen\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-setsecctx* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}setsecctx\ s\ dentry\ ctx\ ctxlen\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-inode-getsecctx* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ \ hook\text{-}inode\text{-}getsecctx\ s\ inode\ ctx\ ctxlen\ \{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

locale *lsm-kernel-hooks* =
fixes $s0 :: 's$
fixes $hook\text{-}kernel\text{-}act\text{-}as :: \ 's \Rightarrow Cred \Rightarrow u32 \Rightarrow \ ('s,\ int)\ nondet\text{-}monad$

fixes *hook-kernel-create-files-as* :: $'s \Rightarrow$ *Cred* $\Rightarrow$ *inode* $\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-kernel-module-request* :: $'s \Rightarrow$ *string* $\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-kernel-load-data* :: $'s \Rightarrow$ *kernel-load-data-id* $\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-kernel-read-file* :: $'s \Rightarrow$ *Files* $\Rightarrow$*kernel-read-file-id* $\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-kernel-post-read-file* :: $'s \Rightarrow$ *Files* $\Rightarrow$ *string* $\Rightarrow$ *nat* $\Rightarrow$*kernel-read-file-id*

$\Rightarrow$ $('s, int)$ *nondet-monad*

assumes *stb-kernel-act-as* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-kernel-act-as s new secid'* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-kernel-create-files-as* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-kernel-create-files-as s c inode* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-kernel-module-request* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-kernel-module-request s name* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-kernel-load-data* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-kernel-load-data s ldataid* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-kernel-read-file*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-kernel-read-file s f rfid* $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-hook-kernel-post-read-file* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$
hook-kernel-post-read-file s file buf size' kid $\{\!|\lambda r\ s.\ \ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

locale *lsm-ipc-hooks* =
fixes *s0* :: $'s$
fixes *ipc-security* :: $'s \Rightarrow$*kern-ipc-perm* $\Rightarrow$ *'ipcsec option*
fixes *msg-security* :: $'s \Rightarrow$*msg-msg* $\Rightarrow$ *'msgsec option*
fixes *hook-ipc-permission* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *int* $\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-ipc-getsecid* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *u32* $\Rightarrow$ $('s, unit)$ *nondet-monad*
fixes *hook-msg-msg-alloc* :: $'s \Rightarrow$ *msg-msg* $\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-msg-msg-free* :: $'s \Rightarrow$ *msg-msg* $\Rightarrow$ $('s, unit)$ *nondet-monad*
fixes *hook-msg-queue-alloc* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-msg-queue-free* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ $('s, unit)$ *nondet-monad*
fixes *hook-msg-queue-associate* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *int*$\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-msg-queue-msgctl* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *IPC-CMD*$\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-msg-queue-msgsnd* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *msg-msg* $\Rightarrow$ *int*$\Rightarrow$ $('s, int)$ *nondet-monad*
fixes *hook-msg-queue-msgrcv* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *msg-msg*$\Rightarrow$ *Task* $\Rightarrow$ *int* $\Rightarrow$ *int*

$\Rightarrow$ $('s, int)$ *nondet-monad*

fixes *hook-shm-alloc* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ $('s, int)$ *nondet-monad*

fixes *hook-shm-free* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ $('s,$ *unit*$)$ *nondet-monad*
fixes *hook-shm-associate* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *int*$\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
fixes *hook-shm-shmctl* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *IPC-CMD*$\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
fixes *hook-shm-shmat* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$*string* $\Rightarrow$*int* $\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
fixes *hook-sem-alloc* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
fixes *hook-sem-free* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ $('s,$ *unit*$)$ *nondet-monad*
fixes *hook-sem-associate*:: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *int*$\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
fixes *hook-sem-semctl* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *IPC-CMD*$\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
fixes *hook-sem-semop* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$*sembuf*$\Rightarrow$*nat* $\Rightarrow$ *int*$\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*

assumes *stb-ipc-permission* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-ipc-permission s ipcp flg* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-ipc-getsecid* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-ipc-getsecid s ipcp secid'* $\{\!|\lambda r\ s.\ r = unit|\!\}$
assumes *stb-msg-msg-alloc* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-msg-msg-alloc s msg* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0|\!\}$
assumes *stb-msg-msg-free* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-msg-msg-free s msg*$\{\!|\lambda r\ s.\ r = unit|\!\}$
assumes *stb-msg-queue-alloc*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-msg-queue-alloc s msq* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-msg-queue-free* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-msg-queue-free s msq*$\{\!|\lambda r\ s.\ r = unit|\!\}$
assumes *stb-msg-queue-associate* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-msg-queue-associate s msq msqflg* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-msg-queue-msgctl*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-msg-queue-msgctl s msq cmd* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-msg-queue-msgsnd* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-msg-queue-msgsnd s msq msg msqflg* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-msg-queue-msgrcv*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-msg-queue-msgrcv s msq msg target type m*
$\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-shm-alloc*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-shm-alloc s shp* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0|\!\}$
assumes *stb-shm-free* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-shm-free s shp*$\{\!|\lambda r\ s.\ r = unit|\!\}$
assumes *stb-shm-associate* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-shm-associate s shp shmflg* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-shm-shmctl*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-shm-shmctl s shp cmd* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-shm-shmat*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-shm-shmat s shp shmaddr shmflg* $\{\!|\lambda r\ s.\ s = sa \wedge (r$

$= 0 \vee r \neq 0)\|\}$

assumes *stb-sem-alloc*:

$\bigwedge sa.\ \{\|\lambda s\ .\ s = sa\ \|\}\ hook\text{-}sem\text{-}alloc\ s\ sma\ \{\|\lambda r\ s.\ r = 0 \vee r \neq 0\|\}$

assumes *stb-sem-free* :

$\bigwedge sa.\ \{\|\lambda s\ .\ s = sa\ \|\}\ hook\text{-}sem\text{-}free\ s\ sma\{\|\lambda r\ s.\ r = unit\|\}$

assumes *stb-sem-associate* :

$\bigwedge sa.\ \{\|\lambda s\ .\ s = sa\ \|\}\ hook\text{-}sem\text{-}associate\ s\ sma\ semflg\ \{\|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\|\}$

assumes *stb-sem-shmctl*:

$\bigwedge sa.\ \{\|\lambda s\ .\ s = sa\ \|\}\ hook\text{-}sem\text{-}semctl\ s\ sma\ cmd\ \{\|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\|\}$

assumes *stb-sem-shmat*:

$\bigwedge sa.\ \{\|\lambda s\ .\ s = sa\ \|\}\ hook\text{-}sem\text{-}semop\ s\ sma\ sops\ nsops\ alter\ \{\|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\|\}$

locale *lsm-other-hooks* =

fixes *s0* :: *'s*

fixes *hook-d-instantiate*:: *'s* ⇒ *dentry* ⇒ *inode option*⇒ (*'s*, *unit*) *nondet-monad*

fixes *hook-getprocattr* :: *'s* ⇒ *Task* ⇒ *string* ⇒ *string* ⇒ (*'s*, *int*) *nondet-monad*

fixes *hook-setprocattr* :: *'s* ⇒ *string* ⇒ *string* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*

fixes *hook-netlink-send* :: *'s* ⇒ *sock* ⇒ *sk-buff*⇒ (*'s*, *int*) *nondet-monad*

fixes *hook-ismaclabel* :: *'s* ⇒ *xattr*⇒ (*'s*, *int*) *nondet-monad*

fixes *hook-secid-to-secctx* :: *'s* ⇒ *u32* ⇒*string* ⇒ *u32* ⇒ (*'s*, *int*) *nondet-monad*

fixes *hook-secctx-to-secid* :: *'s* ⇒ *string* ⇒ *u32*⇒ *u32*⇒ (*'s*, *int*) *nondet-monad*

fixes *hook-release-secctx* :: *'s* ⇒ *string* ⇒ *u32*⇒ (*'s*, *unit*) *nondet-monad*

assumes *stb-d-instantiate* :

$\bigwedge sa.\ \{\|\lambda s\ .\ s = sa\ \|\}\ hook\text{-}d\text{-}instantiate\ sa\ dentry\ inode\ \{\|\lambda r\ s.\ r = unit\|\}$

assumes *stb-getprocattr* :

$\bigwedge sa.\ \{\|\lambda s\ .\ s = sa\ \|\}\ hook\text{-}getprocattr\ sa\ p\ name\ value\ \{\|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\|\}$

assumes *stb-setprocattr* :

$\bigwedge sa.\ \{\|\lambda s\ .\ s = sa\ \|\}\ hook\text{-}setprocattr\ sa\ \ name\ value\ size'\ \{\|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\|\}$

assumes *stb-netlink-send* :

$\bigwedge sa.\ \{\|\lambda s\ .\ s = sa\ \|\}\ hook\text{-}netlink\text{-}send\ s\ sk'\ skb\ \{\|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\|\}$

assumes *stb-ismaclabel* :

$\bigwedge sa.\ \{\|\lambda s.\ s = sa\ \|\}\ hook\text{-}ismaclabel\ s\ name'\ \{\|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\|\}$

assumes *stb-secid-to-secctx*:

$\bigwedge sa.\ \{\|\lambda s.\ s = sa\ \|\}\ hook\text{-}secid\text{-}to\text{-}secctx\ s\ secid'\ secdata\ seclen\ \{\|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\|\}$

assumes *stb-secctx-to-secid* :

$\bigwedge sa.\ \{\|\lambda s.\ s = sa\ \|\}\ hook\text{-}secctx\text{-}to\text{-}secid\ s\ secdata\ seclen\ secid'\ \{\|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\|\}$

assumes *stb-release-secctx*:

$\bigwedge sa.\ \{\|\lambda s.\ s = sa\ \|\}\ hook\text{-}release\text{-}secctx\ s\ secdata\ seclen\ \{\|\lambda r\ s.\ r = unit\|\}$

locale *lsm-network-hooks* =

fixes *s0* :: *'s*
fixes *sk-security* :: *'s* ⇒*sock* ⇒ *'ssec option*
fixes *hook-unix-stream-connect* :: *'s* ⇒ *sock* ⇒ *sock* ⇒ *sock* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-unix-may-send* :: *'s* ⇒ *socket* ⇒ *socket* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-create* :: *'s* ⇒ *Sk-Family* ⇒ *int* ⇒ *int* ⇒*int*⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-post-create* :: *'s* ⇒ *socket* ⇒ *Sk-Family* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-socketpair* :: *'s* ⇒ *socket* ⇒ *socket* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-bind* :: *'s* ⇒ *socket* ⇒ *sockaddr* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-connect* :: *'s* ⇒ *socket* ⇒ *sockaddr* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-listen* :: *'s* ⇒ *socket* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-accept* :: *'s* ⇒ *socket* ⇒ *socket* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-sendmsg* :: *'s* ⇒ *socket* ⇒ *msghdr* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-recvmsg* :: *'s* ⇒ *socket* ⇒ *msghdr* ⇒ *int* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-getsockname* :: *'s* ⇒ *socket* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-getpeername* :: *'s* ⇒ *socket* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-getsockopt* :: *'s* ⇒ *socket* ⇒*int* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-setsockopt* :: *'s* ⇒ *socket* ⇒*int* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-shutdown*:: *'s* ⇒ *socket* ⇒*int* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sock-rcv-skb* :: *'s* ⇒ *sock* ⇒*sk-buff* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-getpeersec-stream* :: *'s* ⇒ *socket* ⇒*string* ⇒*int* ⇒*nat* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-socket-getpeersec-dgram* :: *'s* ⇒ *socket* ⇒*sk-buff option* ⇒ *u32* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sk-alloc* :: *'s* ⇒ *sock*⇒*int* ⇒ *gfp-t* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-sk-free* :: *'s* ⇒ *sock*⇒(*'s*, *unit*) *nondet-monad*
fixes *hook-sk-clone* :: *'s* ⇒ *sock*⇒ *sock* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-sk-classify-flow* :: *'s* ⇒ *sock*⇒ *flowi* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-req-classify-flow* :: *'s* ⇒ *request-sock*⇒ *flowi* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-sock-graft* :: *'s* ⇒ *sock*⇒ *socket* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-inet-conn-request* :: *'s* ⇒ *sock*⇒ *sk-buff* ⇒*request-sock* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-inet-csk-clone* :: *'s* ⇒ *sock* ⇒*request-sock* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-inet-conn-established* :: *'s* ⇒ *sock*⇒ *sk-buff* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-secmark-relabel-packet* :: *'s* ⇒ *u32* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-secmark-refcount-inc* :: *'s* ⇒ (*'s*, *unit*) *nondet-monad*
fixes *hook-secmark-refcount-dec* :: *'s* ⇒ (*'s*, *unit*) *nondet-monad*

fixes *hook-tun-dev-alloc-security* :: *'s* ⇒ *'security* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-tun-dev-free-security* :: *'s* ⇒ *'security* ⇒(*'s*, *unit*) *nondet-monad*
fixes *hook-tun-dev-create* :: *'s* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-tun-dev-attach-queue* :: *'s* ⇒ *'security* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-tun-dev-attach* :: *'s* ⇒ *sock*⇒ *'security*⇒(*'s*, *int*) *nondet-monad*
fixes *hook-tun-dev-open* :: *'s* ⇒ *'security* ⇒(*'s*, *int*) *nondet-monad*

fixes *hook-sctp-assoc-request* :: $'s \Rightarrow$ *sctp-endpoint* $\Rightarrow$ *sk-buff* $\Rightarrow('s,$ *int*) *nondet-monad*
fixes *hook-sctp-bind-connect* :: $'s \Rightarrow$ *sock* $\Rightarrow$ *int* $\Rightarrow$ *sockaddr* $\Rightarrow$*int* $\Rightarrow('s,$ *int*) *nondet-monad*
fixes *hook-sctp-sk-clone* :: $'s \Rightarrow$ *sctp-endpoint* $\Rightarrow$ *sock* $\Rightarrow$ *sock* $\Rightarrow('s,$ *unit*) *nondet-monad*

assumes *stb-unix-stream-connect* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$
hook-unix-stream-connect s sock other newsk
$\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-unix-may-send* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-unix-may-send s sock' other'* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-create* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-create s family type pro kern* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-post-create* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-post-create s sock' family type pro kern*
$\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-socketpair* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-socketpair s socka sockb* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-bind* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-bind s sock' address addrlen* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-connect* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-connect s sock' address addrlen*
$\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-listen* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-listen s sock' backlog* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-accept* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-accept s sock' newsock* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-sendmsg* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-sendmsg s sock' msg size'* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-recvmsg* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-recvmsg s sock' msg size' flgs* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-getsockname* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-getsockname s sock'* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-getpeername* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$ *hook-socket-getpeername s sock'* $\{\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)\}$
assumes *security-socket-getsockopt* :
$\bigwedge sa.\ \{\lambda s\ .\ s = sa\ \}$
hook-socket-getsockopt s sock' level' optname

$\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-socket-setsockopt* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-socket-setsockopt s sock′ level′ optname* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-socket-shutdown*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-socket-shutdown s sock′ how* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-sock-rcv-skb* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-sock-rcv-skb s sock skb* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-socket-getpeersec-stream* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-socket-getpeersec-stream s sock′ optval optlen len′* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-socket-getpeersec-dgram* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$
hook-socket-getpeersec-dgram s sock′ skb′ secid′
$\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-sk-alloc* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-sk-alloc s sk′ family′ priority* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-sk-free* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-sk-free s sock* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-sk-clone* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-sk-clone s sk′ newsk* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-sk-classify-flow* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-sk-classify-flow s sock fl* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-req-classify-flow* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-req-classify-flow s req fl* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-sock-graft* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-sock-graft s sk′ parent′* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-inet-conn-request* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-inet-conn-request s sk′ skb req* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-inet-csk-clone* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-inet-csk-clone s newsk req* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-inet-conn-established* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-inet-conn-established s sk′ skb* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-secmark-relabel-packet* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-secmark-relabel-packet s secid′* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-secmark-refcount-inc* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-secmark-refcount-inc s* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-secmark-refcount-dec* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-secmark-refcount-dec s* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-tun-dev-alloc-security* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-tun-dev-alloc-security s security* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

assumes *security-tun-dev-free-security* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-tun-dev-free-security s security* $\{\!|\lambda r\ s.\ r = unit|\!\}$

assumes *security-tun-dev-create* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-tun-dev-create s* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *security-tun-dev-attach-queue* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-tun-dev-attach-queue s security* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *security-tun-dev-attach* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-tun-dev-attach s sk' security* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *security-tun-dev-open* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-tun-dev-open s security* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *security-sctp-assoc-request* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-sctp-assoc-request s ep skb* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *security-sctp-bind-connect* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-sctp-bind-connect s sk' optname address addrlen*
$\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *security-sctp-sk-clone* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-sctp-sk-clone s ep sk' newsk* $\{\!|\lambda r\ s.\ r = unit|\!\}$

locale *lsm-infiniband-hooks* =

fixes *s0* :: *'s*
fixes *hook-ib-pkey-access* :: *'s* ⇒ *'v* ⇒ *nat* ⇒ *nat* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-ib-endport-manage-subnet*:: *'s* ⇒ *'v* ⇒ *string* ⇒ *nat* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-ib-alloc-security* :: *'s* ⇒ *'v list* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-ib-free-security* :: *'s* ⇒ *'v list* ⇒ (*'s*, *unit*) *nondet-monad*
assumes *stb-ib-pkey-access* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-ib-pkey-access sa sec prefix' pkey* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-ib-endport-manage-subnet* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-ib-endport-manage-subnet sa sec dev-name prot-num* $\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-ib-alloc-security* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-ib-alloc-security sa sec'* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0|\!\}$
assumes *stb-ib-free-security* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-ib-free-security sa sec'* $\{\!|\lambda r\ s.\ r = unit|\!\}$

locale *lsm-network-xfrm-hooks* =

fixes *s0* :: *'s*
fixes *hook-xfrm-policy-alloc* :: *'s* ⇒*xfrm-sec-ctx* ⇒ *xfrm-user-sec-ctx* ⇒ *gfp-t* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-xfrm-policy-clone* :: *'s* ⇒*xfrm-sec-ctx* ⇒ *xfrm-user-sec-ctx* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-xfrm-policy-free* :: *'s* ⇒*xfrm-sec-ctx* ⇒(*'s*, *unit*) *nondet-monad*
fixes *hook-xfrm-policy-delete* :: *'s* ⇒*xfrm-sec-ctx* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-xfrm-state-alloc* :: *'s* ⇒*xfrm-state* ⇒ *xfrm-sec-ctx* ⇒(*'s*, *int*) *nondet-monad*

fixes *hook-xfrm-state-alloc-acquire* :: *'s* ⇒*xfrm-state* ⇒ *xfrm-sec-ctx* => *u32* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-xfrm-state-delete* :: *'s* ⇒*xfrm-state* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-xfrm-state-free* :: *'s* ⇒*xfrm-state* ⇒(*'s*, *unit*) *nondet-monad*
fixes *hook-xfrm-policy-lookup* :: *'s* ⇒*xfrm-sec-ctx* ⇒ *u32* ⇒ *u8* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-xfrm-state-pol-flow-match* :: *'s* ⇒*xfrm-state* ⇒*xfrm-policy* ⇒ *flowi* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-xfrm-decode-session* :: *'s* ⇒*sk-buff* ⇒ *u32* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-skb-classify-flow* :: *'s* ⇒*sk-buff* ⇒ *flowi* ⇒(*'s*, *unit*) *nondet-monad*
assumes *stb-xfrm-policy-alloc* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-policy-alloc sa ctxp sec-ctx gfp'* ⦃$\lambda r\ s.\ r = 0 \lor r \neq 0$⦄
assumes *stb-xfrm-policy-clone* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-policy-clone sa old-ctx new-ctxp* ⦃$\lambda r\ s.\ r = 0 \lor r \neq 0$⦄
assumes *stb-xfrm-policy-free* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-policy-free sa ctx* ⦃$\lambda r\ s.\ r = unit$⦄
assumes *stb-xfrm-policy-delete* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-policy-delete sa ctx* ⦃$\lambda r\ s.\ r = 0 \lor r \neq 0$⦄
assumes *stb-xfrm-state-alloc* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-state-alloc sa x sec-ctx'* ⦃$\lambda r\ s.\ r = 0 \lor r \neq 0$⦄
assumes *stb-xfrm-state-alloc-acquire* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-state-alloc-acquire sa x plosec secid'* ⦃$\lambda r\ s.\ r = 0 \lor r \neq 0$⦄
assumes *stb-xfrm-state-delete* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-state-delete sa x* ⦃$\lambda r\ s.\ r = 0 \lor r \neq 0$⦄
assumes *stb-xfrm-state-free* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-state-free sa x* ⦃$\lambda r\ s.\ r = unit$⦄
assumes *stb-xfrm-policy-lookup* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-policy-lookup sa ctx fl-secid dir* ⦃$\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)$⦄
assumes *stb-xfrm-state-pol-flow-match* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-state-pol-flow-match sa x xp fl* ⦃$\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)$⦄
assumes *stb-xfrm-decode-session* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-xfrm-decode-session sa skb secid'* ⦃$\lambda r\ s.\ r = 0 \lor r \neq 0$⦄
assumes *stb-skb-classify-flow* :
$\bigwedge sa.$ ⦃λs . $s = sa$ ⦄ *hook-skb-classify-flow sa skb fl* ⦃$\lambda r\ s.\ r = unit$⦄

locale *lsm-path-hooks* =

fixes *s0* :: *'s*
fixes *hook-path-unlink* :: *'s* ⇒ *path* ⇒ *dentry* ⇒ (*'s*, *int*) *nondet-monad*
fixes *hook-path-mkdir* :: *'s* ⇒ *path* ⇒ *dentry* ⇒*nat* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-path-rmdir* :: *'s* ⇒ *path* ⇒ *dentry* ⇒(*'s*, *int*) *nondet-monad*
fixes *hook-path-mknod* :: *'s* ⇒ *path* ⇒ *dentry* ⇒ *nat* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*

fixes *hook-path-truncate* :: $'s \Rightarrow path \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-path-symlink* :: $'s \Rightarrow path \Rightarrow dentry \Rightarrow string \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-path-link* :: $'s \Rightarrow dentry \Rightarrow path \Rightarrow dentry \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-path-rename* :: $'s \Rightarrow path \Rightarrow dentry \Rightarrow path \Rightarrow dentry \Rightarrow ('s,\ int)$ $nondet\text{-}monad$
fixes *hook-path-chmod* :: $'s \Rightarrow path \Rightarrow nat \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-path-chown* :: $'s \Rightarrow path \Rightarrow kuid\text{-}t \Rightarrow kgid\text{-}t \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-path-chroot* :: $'s \Rightarrow path \Rightarrow ('s,\ int)\ nondet\text{-}monad$
assumes *stb-path-unlink* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}unlink\ s\ dir\ dentry\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-mkdir* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}mkdir\ s\ dir\ dentry\ m\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-rmdir* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}rmdir\ s\ dir\ dentry\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-mknod* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}mknod\ s\ dir\ dentry\ m\ dev\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-truncate* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}truncate\ s\ dir\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-symlink* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}symlink\ s\ dir\ dentry\ old\text{-}name\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-link* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}link\ s\ old\text{-}dentry\ new\text{-}dir\ new\text{-}dentry$
$\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-rename* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}rename\ s\ old\text{-}dir\ old\text{-}dentry\ new\text{-}dir\ new\text{-}dentry$

$\{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-chmod* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}chmod\ s\ path\ m\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-chown* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}chown\ s\ path\ uid'\ gid'\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$
assumes *stb-path-chroot* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}\ hook\text{-}path\text{-}chroot\ s\ path\ \{\!|\lambda r\ s.\ s = sa \wedge (r = 0 \vee r \neq 0)|\!\}$

locale *lsm-key-hooks* =

fixes *s0* :: $'s$
fixes *key-security* :: $'s \Rightarrow key \Rightarrow 'ksec\ option$
fixes *hook-key-alloc* :: $'s \Rightarrow key \Rightarrow Cred \Rightarrow nat \Rightarrow ('s,\ int)\ nondet\text{-}monad$
fixes *hook-key-free* :: $'s \Rightarrow key \Rightarrow ('s,\ unit)\ nondet\text{-}monad$

fixes *hook-key-permission* :: $'s \Rightarrow$ *key-ref-t* $\Rightarrow$ *Cred* $\Rightarrow$ *nat* $\Rightarrow$($'s$, *int*) *nondet-monad*
fixes *hook-key-getsecurity* :: $'s \Rightarrow$ *key* $\Rightarrow$ *string* $\Rightarrow$($'s$, *int*) *nondet-monad*

assumes *stb-key-alloc*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-key-alloc sa k cred' flag* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-key-free* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-key-free sa k* $\{\!|\lambda r\ s.\ r = unit \land key\text{-}security\ s\ k =$ *None* $|\!\}$
assumes *stb-key-permission*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-key-permission sa key-ref c perm* $\{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$
assumes *stb-key-getsecurity*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-key-getsecurity sa key' buffer* $\{\!|\lambda r\ s.\ s = sa \land (r = 0 \lor r \neq 0)|\!\}$

locale *lsm-audit-hooks* =

fixes *s0* :: $'s$
fixes *hook-audit-rule-init* :: $'s \Rightarrow$ *nat* $\Rightarrow$ *enum-audit* $\Rightarrow$ *string* $\Rightarrow$ *string* $\Rightarrow$ ($'s$, *int*) *nondet-monad*
fixes *hook-audit-rule-known* :: $'s \Rightarrow$ *audit-krule* $\Rightarrow$ ($'s$, *int*) *nondet-monad*
fixes *hook-audit-rule-match* :: $'s \Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *enum-audit* $\Rightarrow$ *string* $\Rightarrow$ *audit-context* $\Rightarrow$ ($'s$, *int*) *nondet-monad*
fixes *hook-audit-rule-free* :: $'s \Rightarrow$ *string* $\Rightarrow$ ($'s$, *unit*) *nondet-monad*

assumes *stb-audit-rule-init*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-audit-rule-init s field op rulestr vrule* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-audit-rule-known* :
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-audit-rule-known s krule* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-audit-rule-match*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-audit-rule-match s secid' field op vrule actx* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-key-audit-rule-free*:
$\bigwedge sa.$ $\{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-audit-rule-free s lsmrule* $\{\!|\lambda r\ s.\ r = unit|\!\}$

locale *lsm-bpf-hooks* =

fixes *s0* :: $'s$
fixes *hook-bpf* :: $'s \Rightarrow$ *int* $\Rightarrow$ *bpf-attr* $\Rightarrow$ *nat* $\Rightarrow$($'s$, *int*) *nondet-monad*
fixes *hook-bpf-map* :: $'s \Rightarrow$ *bpf-map* $\Rightarrow$ *mode* $\Rightarrow$($'s$, *int*) *nondet-monad*
fixes *hook-bpf-prog* :: $'s \Rightarrow$ *bpf-prog* $\Rightarrow$($'s$, *int*) *nondet-monad*
fixes *hook-bpf-map-alloc* :: $'s \Rightarrow$ *bpf-map* $\Rightarrow$($'s$, *int*) *nondet-monad*
fixes *hook-bpf-map-free* :: $'s \Rightarrow$ *bpf-map* $\Rightarrow$($'s$, *unit*) *nondet-monad*
fixes *hook-bpf-prog-alloc* :: $'s \Rightarrow$ *bpf-prog-aux* $\Rightarrow$($'s$, *int*) *nondet-monad*
fixes *hook-bpf-prog-free* :: $'s \Rightarrow$ *bpf-prog-aux* $\Rightarrow$($'s$, *unit*) *nondet-monad*
assumes *stb-bpf* :

$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-bpf sa cmd attr′ size′* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-bpf-map* :
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-bpf-map sa bmap fmode* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-bpf-prog*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-bpf-prog sa prog* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-bpf-map-alloc*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-bpf-map-alloc sa bmap* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-bpf-map-free*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-bpf-map-free sa bmap* $\{\!|\lambda r\ s.\ r = unit|\!\}$
assumes *stb-bpf-prog-alloc*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-bpf-prog-alloc sa proga*$\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
assumes *stb-bpf-prog-free*:
$\bigwedge sa.\ \{\!|\lambda s\ .\ s = sa\ |\!\}$ *hook-bpf-prog-free sa proga* $\{\!|\lambda r\ s.\ r = unit|\!\}$

locale *lsm-hooks* =
lsm-superblock-hooks s0 +
lsm-task-hooks s0 +
lsm-binder-hooks s0 +
lsm-ptrace-hooks s0 +
lsm-capable-hooks s0 +
lsm-bprm-hooks s0 +
lsm-dentry-hooks s0 +
lsm-inode-hooks s0 +
lsm-file-hooks s0+
lsm-kernel-hooks s0 +
lsm-ipc-hooks s0 +
lsm-other-hooks s0 +
lsm-network-hooks s0 +
lsm-infiniband-hooks s0 +
lsm-network-xfrm-hooks s0 +
lsm-path-hooks s0 +
lsm-key-hooks s0 +
lsm-audit-hooks s0 +
lsm-bpf-hooks s0
for *s0* :: *′s*

begin
end
end

27 LSM Model

theory *Linux-LSM-Model*
imports
SOAC
LSM-Cap
Linux-LSM-Hooks
../lib/Monad-WP/NonDetMonadVCG

begin

In this theory, we introduce LSM Model

27.1 def security opts type

definition *security-init-mnt-opts* ≡ (|*mnt-opts* = [],*mnt-opts-flags* = [],*num-mnt-opts*=*0*|)

27.2 lsm model

locale *lsm* = *lsm-hooks state* + *SOModel subj-label obj-label access-rules Subj Obj request*
  for *state* :: 's
  and *subj-label* :: 's ⇒*Subj* ⇒ *subject-label*
  and *obj-label* :: 's =>*Obj* ⇒ *object-label*
  and *access-rules* :: *Label* ⇒ *Label* ⇒ *access set*
  and *Subj* :: 's ⇒ *Subj set*
  and *Obj* :: 's ⇒ *Obj set*
  and *request* :: 's ⇒*Subj* ⇒ *Obj* ⇒ *Request*⇒ *decision*
  +
  fixes *k-task* :: 's ⇒ *process-id* ⇀ *Task*
  fixes *inodes* :: 's ⇒ *inum* ⇀ *inode*

begin

definition *security-capget* :: 's ⇒ *Task* ⇒ *kct* ⇒ *kct* ⇒*kct* ⇒ ('s, *int*) *nondet-monad*
  where *security-capget s target effective inheritable permitted* ≡
      hook-capget s target effective inheritable permitted

definition *security-capset* :: 's ⇒ *Cred* ⇒ *Cred* ⇒ *kct* ⇒ *kct* ⇒*kct* ⇒ ('s, *int*) *nondet-monad*
  where *security-capset s new old effective inheritable permitted* ≡
      hook-capset s new old effective inheritable permitted

definition *security-capable* :: 's ⇒ *Cred* ⇒ *ns* ⇒ *cap* ⇒ ('s, *int*) *nondet-monad*
  where *security-capable s c ns cap* ≡ *hook-capable s c ns cap*

definition *security-capable-noaudit* :: 's ⇒ *Cred* ⇒ *ns* ⇒ *cap* ⇒ ('s, *int*) *nondet-monad*
  where *security-capable-noaudit s c ns cap* ≡ *hook-capable-noaudit s c ns cap*

definition *security-quotactl*:: 's ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *super-block option* ⇒ ('s, *int*) *nondet-monad*
  where *security-quotactl s cmds t id' sb* ≡ *hook-quotactl s cmds t id' sb*

definition *security-quota-on*:: 's ⇒ *dentry*⇒ ('s, *int*) *nondet-monad*
  where *security-quota-on s dentry* ≡ *hook-quota-on s dentry*

definition *security-settime64* :: 's ⇒ *ts* ⇒ *tz option* ⇒ ('s, *int*) *nondet-monad*

where *security-settime64 s ts tz* ≡ *hook-settime64 s ts tz*

definition *vm-enough-memory* :: $'s \Rightarrow mm \Rightarrow pages \Rightarrow int \Rightarrow int$
where *vm-enough-memory s mm' p pages* ≡ *0*

definition *security-vm-enough-memory-mm* :: $'s \Rightarrow mm \Rightarrow pages \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-vm-enough-memory-mm s mm' pages* ≡ *do*
rc ← *hook-vm-enough-memory-mm s mm' pages*;
cap-sys-admin ← (*if rc* ≤ *0 then return 0*
else return 1
);
return(*vm-enough-memory s mm' pages cap-sys-admin*)
od

definition *security-binder-set-context-mgr* :: $'s \Rightarrow Task \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-binder-set-context-mgr s mgr* ≡ *hook-binder-set-context-mgr s mgr*

definition *security-binder-transaction* :: $'s \Rightarrow Task \Rightarrow Task \Rightarrow ('s,\ int)$ *nondet-monad*
where*security-binder-transaction s from to* ≡ *hook-binder-transaction s from to*

definition *security-binder-transfer-binder* :: $'s \Rightarrow Task \Rightarrow Task \Rightarrow ('s,\ int)$ *nondet-monad*
where*security-binder-transfer-binder s from to* ≡ *hook-binder-transfer-binder s from to*

definition *security-binder-transfer-file* :: $'s \Rightarrow Task \Rightarrow Task \Rightarrow Files \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-binder-transfer-file s from to file* ≡ *hook-binder-transfer-file s from to file*

definition *security-ptrace-access-check* :: $'s \Rightarrow Task \Rightarrow nat \Rightarrow ('s,\ int)$ *nondet-monad*

where *security-ptrace-access-check s child m* ≡ *hook-ptrace-access-check s child m*

definition *security-ptrace-traceme* :: $'s \Rightarrow Task \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-ptrace-traceme s parent'* ≡ *hook-ptrace-traceme s parent'*

definition *security-syslog* :: $'s \Rightarrow int \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-syslog s type* ≡ *hook-syslog s type*

definition *ima-bprm-check bprm* = *0*

definition *security-bprm-set-creds* :: $'s \Rightarrow linux\text{-}binprm \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-bprm-set-creds s bprm* ≡ *hook-bprm-set-creds s bprm*

definition *security-bprm-check* :: $'s \Rightarrow$*linux-binprm* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-bprm-check s bprm* $\equiv$ *do*
ret $\leftarrow$ *hook-bprm-check s bprm*;
rc $\leftarrow$ (*if ret* $\neq$ *0 then return ret*
else return (*ima-bprm-check bprm*));
return rc
od

definition *security-bprm-committing-creds* :: $'s \Rightarrow$ *linux-binprm* $\Rightarrow$ $('s,\ unit)$ *nondet-monad*
where *security-bprm-committing-creds s bprm* $\equiv$ *hook-bprm-committing-creds s bprm*

definition *security-bprm-committed-creds* :: $'s \Rightarrow$ *linux-binprm* $\Rightarrow$ $('s,\ unit)$ *nondet-monad*
where *security-bprm-committed-creds s bprm* $\equiv$ *hook-bprm-committed-creds s bprm*

definition *security-sb-alloc* :: $'s \Rightarrow$ *super-block* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-sb-alloc s sb* $\equiv$ *hook-sb-alloc s sb*

definition *security-sb-free* :: $'s \Rightarrow$ *super-block* $\Rightarrow$ $('s,\ unit)$ *nondet-monad*
where *security-sb-free s sb* $\equiv$ *hook-sb-free s sb*

definition *security-sb-copy-data* :: $'s \Rightarrow$ *string* $\Rightarrow$ *string* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-sb-copy-data s orig copy* $\equiv$ *hook-sb-copy-data s orig copy*

definition *security-sb-remount* :: $'s \Rightarrow$*super-block* $\Rightarrow$ *Void* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-sb-remount s sb data* $\equiv$ *hook-sb-remount s sb data*

definition *security-sb-kern-mount* :: $'s \Rightarrow$*super-block* $\Rightarrow$ *int* $\Rightarrow$ *string* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-sb-kern-mount s sb flgs data* $\equiv$ *hook-sb-kern-mount s sb flgs data*

definition *security-sb-show-options* :: $'s \Rightarrow$ *seq-file* $\Rightarrow$ *super-block* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-sb-show-options s m sb* $\equiv$ *hook-sb-show-options s m sb*

definition *security-sb-statfs* :: $'s \Rightarrow$ *dentry* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-sb-statfs s dentry* $\equiv$ *hook-sb-statfs s dentry*

definition *security-sb-mount* :: $'s \Rightarrow$ *string* $\Rightarrow$ *path* $\Rightarrow$ *string* $\Rightarrow$ *int* $\Rightarrow$ *Void* $\Rightarrow$$('s,\ int)$ *nondet-monad*
where *security-sb-mount s dev-name path type flgs data* $\equiv$
hook-sb-mount s dev-name path type flgs data

definition *security-sb-umount* :: $'s \Rightarrow$*vfsmount* $\Rightarrow$ *int* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-sb-umount s vmnt flgs* $\equiv$ *hook-sb-umount s vmnt flgs*

definition *security-sb-pivotroot* :: $'s \Rightarrow path \Rightarrow path \Rightarrow ('s, int)$ *nondet-monad*
where *security-sb-pivotroot s old-path new-path* $\equiv$ *hook-sb-pivotroot s old-path new-path*

definition *security-sb-set-mnt-opts* :: $'s \Rightarrow super\text{-}block \Rightarrow opts \Rightarrow nat \Rightarrow nat \Rightarrow ('s, int)$ *nondet-monad*
where *security-sb-set-mnt-opts s sb opt kern-flags set-kern-flags* $\equiv$
hook-sb-set-mnt-opts s sb opt kern-flags set-kern-flags

definition *security-sb-clone-mnt-opts* :: $'s \Rightarrow super\text{-}block \Rightarrow super\text{-}block \Rightarrow int \Rightarrow int \Rightarrow ('s, int)$ *nondet-monad*
where *security-sb-clone-mnt-opts s oldsb newsb kern-flags set-kern-flags* $\equiv$
hook-sb-clone-mnt-opts s oldsb newsb kern-flags set-kern-flags

definition *security-sb-parse-opts-str* :: $'s \Rightarrow string \Rightarrow opts \Rightarrow ('s, int)$ *nondet-monad*
where *security-sb-parse-opts-str s options opt* $\equiv$ *hook-sb-parse-opts-str s options opt*

definition *d-backing-inode* :: $'s \Rightarrow dentry \Rightarrow$ *inode option*
where *d-backing-inode s upper* $\equiv$((*inodes s*)(*d-inode upper*))

definition *integrity-inode-free* :: $'s \Rightarrow inode \Rightarrow ('s, unit)$ *nondet-monad*
where *integrity-inode-free s inode* $\equiv$ *return*()

definition *security-inode-alloc* :: $'s \Rightarrow inode \Rightarrow ('s, int)$ *nondet-monad*
where *security-inode-alloc s inode* $\equiv$ *hook-inode-alloc s inode*

definition *security-inode-free* :: $'s \Rightarrow inode \Rightarrow ('s, unit)$ *nondet-monad*
where *security-inode-free s inode* $\equiv$ *do*
integrity-inode-free s inode;
hook-inode-free s inode
od

definition *evm-inode-init-security* :: $inode \Rightarrow xattrs \Rightarrow xattrs \Rightarrow int$
where *evm-inode-init-security inode xattr-array evm* $\equiv$ *0*

definition *initxattrss* :: $inode \Rightarrow xattrs\ list \Rightarrow string \Rightarrow int$
where *initxattrss inode xattr-array fs-data* $\equiv$ *1*

definition *security-inode-init-security* :: $'s \Rightarrow inode \Rightarrow inode \Rightarrow string \Rightarrow initxattrs$
$\Rightarrow string \Rightarrow ('s, int)$ *nondet-monad*
where *security-inode-init-security s inode dir qstr initxattrs' fsdata = do*
new-xattrs $\leftarrow$ *return*(*SOME x*::*xattrs list*. *True*);
lsm-xattr $\leftarrow$ *return*(*SOME x*::*xattrs*. *True*);
evm-xattr $\leftarrow$ *return*(*SOME x*::*xattrs*. *True*);

xattr ← *return*(*SOME x*::*xattr*. *True*);
rc ← (*if unlikely* (*IS-PRIVATE inode*) *then return* (*0*)
else
if initxattrs′ = *0 then*
(*hook-inode-init-security s inode dir qstr* ′′′′ ′′′′ *0*)
else do
lsm-xattrs ← *return* (*new-xattrs*);
lsm-xattr ← *return* (*lsm-xattrs* ! *0*);
ret ← (*hook-inode-init-security s inode dir qstr* (*xattr-name*
lsm-xattr)
(*xattr-value lsm-xattr*) (*xattr-value-len*
lsm-xattr));
if ret ≠ *0 then*
if ret = (−*EOPNOTSUPP*) *then*
return 0
else return ret
else do
evm-xattr ← *return*(*lsm-xattrs* ! *1*);
ret ← *return*(*evm-inode-init-security inode lsm-xattr*
evm-xattr);
if ret ≠ *0 then*
if ret = (−*EOPNOTSUPP*) *then*
(*return 0*)
else return ret
else
do
ret ← *return* (*initxattrss inode new-xattrs*
fsdata);
if ret = (−*EOPNOTSUPP*) *then* (
return 0)
else return ret
od
od
od

);
return rc
od

definition *security-old-inode-init-security* :: ′*s* ⇒ *inode* ⇒ *inode* ⇒ *qstr* ⇒ *string*

⇒ *string* ⇒ *int* ⇒ (′*s*, *int*) *nondet-monad*
where *security-old-inode-init-security s inode dir qstr name value len′* ≡
hook-old-inode-init-security s inode dir qstr name value len′

definition *security-inode-create* :: ′*s* ⇒ *inode* ⇒ *dentry*⇒ *mode* ⇒ (′*s*, *int*) *nondet-monad*
where *security-inode-create s dir dentry m* ≡
if unlikely (*IS-PRIVATE dir*) *then*

return 0
else
hook-inode-create s dir dentry m

definition *security-inode-link* :: $'s \Rightarrow$ *dentry* $\Rightarrow$ *inode* $\Rightarrow$ *dentry* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-inode-link s old-dentry dir new-dentry* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s old-dentry*))) *then*
return 0
else
hook-inode-link s old-dentry dir new-dentry

definition *security-inode-unlink* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *dentry* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-inode-unlink s dir dentry* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s dentry*))) *then*
return 0
else
hook-inode-unlink s dir dentry

definition *security-inode-symlink* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *dentry* $\Rightarrow$ *string* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-inode-symlink s dir dentry old-name* $\equiv$
if unlikely (*IS-PRIVATE dir*) *then*
return 0
else
hook-inode-symlink s dir dentry old-name

definition *security-inode-mkdir* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *dentry* $\Rightarrow$ *mode* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-inode-mkdir s dir dentry m* $\equiv$
if unlikely (*IS-PRIVATE dir*) *then*
return 0
else
hook-inode-mkdir s dir dentry m

definition *security-inode-rmdir* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *dentry* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-inode-rmdir s dir dentry* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s dentry*))) *then*
return 0
else
hook-inode-rmdir s dir dentry

definition *security-inode-mknod* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *dentry* $\Rightarrow$ *mode* $\Rightarrow$ *dev-t* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-inode-mknod s dir dentry m dev* $\equiv$
if unlikely (*IS-PRIVATE dir*) *then*
return 0
else
hook-inode-mknod s dir dentry m dev

definition *security-inode-rename* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *dentry* $\Rightarrow$ *inode* $\Rightarrow$ *dentry* $\Rightarrow$

flags
$\Rightarrow$ ('s, *int*) *nondet-monad*
where *security-inode-rename s old-dir old-dentry new-dir new-dentry flgs* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s old-dentry*))) $\vee$
((*d-is-positive new-dentry*) $\wedge$ *IS-PRIVATE* (*the*(*d-backing-inode s old-dentry*)) $\neq$ *0*)
then return 0
else if ((*int flgs*) *AND RENAME-EXCHANGE*) $\neq$ *0 then*
do
err $\leftarrow$ (*hook-inode-rename s new-dir new-dentry old-dir old-dentry*);
if err $\neq$ *0 then*
return err
else (*hook-inode-rename s old-dir old-dentry new-dir new-dentry*)
od
else
(*hook-inode-rename s old-dir old-dentry new-dir new-dentry*)

definition *security-inode-readlink* :: 's $\Rightarrow$ *dentry* $\Rightarrow$ ('s, *int*) *nondet-monad*
where *security-inode-readlink s dentry* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s dentry*))) *then*
return 0
else
hook-inode-readlink s dentry

definition *security-inode-follow-link* :: 's $\Rightarrow$ *dentry* $\Rightarrow$ *inode* $\Rightarrow$ *bool* $\Rightarrow$ ('s, *int*) *nondet-monad*
where *security-inode-follow-link s dentry inode rcu'* $\equiv$
if unlikely (*IS-PRIVATE inode*) *then*
return 0
else
hook-inode-follow-link s dentry inode rcu'

definition *security-inode-permission* :: 's $\Rightarrow$ *inode* $\Rightarrow$ *mask* $\Rightarrow$ ('s, *int*) *nondet-monad*
where *security-inode-permission s inode m* $\equiv$
if unlikely (*IS-PRIVATE inode*) *then*
return 0
else
hook-inode-permission s inode m

definition *evm-inode-setattr* :: 's $\Rightarrow$ *dentry* $\Rightarrow$ *iattr* $\Rightarrow$ ('s, *int*) *nondet-monad*
where *evm-inode-setattr s dentry at* $\equiv$ *return 0*

definition *security-inode-setattr* :: 's $\Rightarrow$ *dentry* $\Rightarrow$ *iattr* $\Rightarrow$ ('s, *int*) *nondet-monad*
where *security-inode-setattr s dentry attr'* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s dentry*))) *then*
return 0
else do
ret $\leftarrow$ *hook-inode-setattr s dentry attr'*;

if ret $\neq$ *0 then*
return ret
else
evm-inode-setattr s dentry attr'
od

definition *security-inode-getattr* :: *'s* $\Rightarrow$ *path* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-getattr s path* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*p-dentry path*))))
then
return 0
else
hook-inode-getattr s path

definition *ima-inode-setxattr* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$*xattr* $\Rightarrow$*string* $\Rightarrow$*int* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *ima-inode-setxattr s d x value flg* $\equiv$ *return 0*

definition *evm-inode-setxattr* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$*xattr* $\Rightarrow$*string* $\Rightarrow$*int* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *evm-inode-setxattr s d x value flg* $\equiv$ *return 0*

definition *security-inode-setxattr* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$ *xattr* $\Rightarrow$ *string* $\Rightarrow$ *int* $\Rightarrow$ *flags*

$\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-setxattr s dentry name value size' flgs* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s dentry*)))
then return 0
else do
ret $\leftarrow$ *hook-inode-setxattr s dentry name value size' flgs*;
if ret $\neq$ *1 then*
cap-inode-setxattr s dentry name value size' flgs
else if ret $\neq$ *0 then*
return ret
else
do
ret $\leftarrow$ *ima-inode-setxattr s dentry name value size'*;
if ret $\neq$ *0 then*
return ret
else
evm-inode-setxattr s dentry name value size'
od
od

definition *evm-inode-post-setxattr* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$*xattr* $\Rightarrow$*string* $\Rightarrow$*int* $\Rightarrow$ (*'s*, *unit*) *nondet-monad*
where *evm-inode-post-setxattr s d x value flg* $\equiv$ *return* ()

definition *security-inode-post-setxattr* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$ *xattr* $\Rightarrow$ *string* $\Rightarrow$ *int* $\Rightarrow$

flags

$\Rightarrow$ (*'s*, *unit*) *nondet-monad*

where *security-inode-post-setxattr s dentry name value size' flgs* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s dentry*))) *then*
return ()
else
do
hook-inode-post-setxattr s dentry name value size' flgs;
evm-inode-post-setxattr s dentry name value size'
od

definition *security-inode-getxattr* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$ *xattr* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-getxattr s dentry name* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s dentry*)))
then
return 0
else
hook-inode-getxattr s dentry name

definition *security-inode-listxattr* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-listxattr s dentry* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s dentry*)))
then
return 0
else
hook-inode-listxattr s dentry

definition *current-user-ns* :: *'s* $\Rightarrow$ *ns*
where *current-user-ns s* = *user-ns* (*cred*(*the*((*k-task s*) (*current s*))))

definition *privileged-wrt-inode-uidgid* :: *ns* $\Rightarrow$ *inode* $\Rightarrow$ *bool*
where *privileged-wrt-inode-uidgid ns inode* $\equiv$
(*kuid-has-mapping ns* (*i-uid inode*))
$\wedge$ (*kgid-has-mapping ns* (*i-gid inode*))

definition *capable-wrt-inode-uidgid* :: *'s* $\Rightarrow$ *inode* $\Rightarrow$ *int*$\Rightarrow$*bool*
where *capable-wrt-inode-uidgid s inode cap* $\equiv$
let ns = *current-user-ns s*
in (*ns-capable ns cap*) $\wedge$ *privileged-wrt-inode-uidgid ns inode*

definition *cap-inode-removexattr* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$*xattr* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *cap-inode-removexattr s dentry name* $\equiv$ *do*
ns $\leftarrow$ *return* (*s-user-ns* (*d-sb dentry*));
rc $\leftarrow$ (*if name* = *XATTR-SECURITY-PREFIX*
then
return 0
else
if name = *XATTR-NAME-CAPS then*

do
inode ← *return* ((*d-backing-inode s dentry*));
if inode = *None then*
return(−*EINVAL*)
else
if ¬(*capable-wrt-inode-uidgid s* (*the inode*) *CAP-SETFCAP*)
then
return(−*EPERM*)
else
return 0
od
else
if ¬(*ns-capable ns CAP-SYS-ADMIN*)
then
return (−*EPERM*)
else
return 0
);
return(*rc*)
od

definition *ima-inode-removexattr* :: *dentry* ⇒ *xattr* ⇒ *int*
where *ima-inode-removexattr dentry name* ≡ *0*

definition *evm-inode-removexattr* :: *dentry* ⇒ *xattr* ⇒ *int*
where *evm-inode-removexattr dentry name* ≡ *0*

definition *security-inode-removexattr* :: *'s* ⇒ *dentry* ⇒ *xattr* ⇒ (*'s*, *int*) *nondet-monad*

where *security-inode-removexattr s dentry name* ≡
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s dentry*))) *then return 0*
else do
ret ← *hook-inode-removexattr s dentry name*;
rc ←*if ret* = *1 then*
cap-inode-removexattr s dentry name
else if ret ≠ *0 then*
return ret
else do
ret ← *return*(*ima-inode-removexattr dentry name*);
if ret ≠ *0 then*
return ret
else
return(*evm-inode-removexattr dentry name*)
od ;
return rc
od

definition *security-inode-need-killpriv* :: *'s* ⇒ *dentry* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-need-killpriv s dentry* ≡ *hook-inode-need-killpriv s dentry*

definition *security-inode-killpriv* :: $'s \Rightarrow$ *dentry* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-inode-killpriv s dentry* $\equiv$ *hook-inode-killpriv s dentry*

definition *security-inode-getsecurity* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *xattr* $\Rightarrow$ *Void* $\Rightarrow$ *bool* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-inode-getsecurity s inode name buffer alloc* $\equiv$
if unlikely (*IS-PRIVATE* (*inode*)) *then return* $(-EOPNOTSUPP)$
else do
rc $\leftarrow$ *hook-inode-getsecurity s inode name buffer alloc*;
if rc $\neq$ $(-EOPNOTSUPP)$
then
return rc
else
return$(-EOPNOTSUPP)$
od

definition *security-inode-setsecurity* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *xattr* $\Rightarrow$ *Void* $\Rightarrow$ *nat* $\Rightarrow$ *int* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-inode-setsecurity s inode name value size' flgs* $\equiv$
if unlikely (*IS-PRIVATE* (*inode*))
then
return $(-EOPNOTSUPP)$
else do
rc $\leftarrow$ *hook-inode-setsecurity s inode name value size' flgs* ;
if rc $\neq$ $(-EOPNOTSUPP)$
then
return rc
else
return$(-EOPNOTSUPP)$
od

definition *security-inode-listsecurity* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *Void* $\Rightarrow$ *int* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-inode-listsecurity s inode buffer bsize* $\equiv$
if unlikely (*IS-PRIVATE* (*inode*))
then
return 0
else
hook-inode-listsecurity s inode buffer bsize

definition *security-inode-getsecid* :: $'s \Rightarrow$ *inode* $\Rightarrow$ *u32* $\Rightarrow$ $('s,\ unit)$ *nondet-monad*
where *security-inode-getsecid s inode secid'* $\equiv$ *hook-inode-getsecid s inode secid'*

definition *security-inode-copy-up* :: $'s \Rightarrow$ *dentry* $\Rightarrow$ *Cred option* $\Rightarrow$ $('s,\ int)$ *nondet-monad*
where *security-inode-copy-up s src new* $\equiv$ *hook-inode-copy-up s src new*

definition *security-inode-copy-up-xattr* :: $'s \Rightarrow$ *xattr* $\Rightarrow$ $('s,\ int)$ *nondet-monad*

where *security-inode-copy-up-xattr s name* ≡ *hook-inode-copy-up-xattr s name*

definition *security-inode-invalidate-secctx* :: *'s* ⇒ *inode* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-inode-invalidate-secctx s inode* ≡ *hook-inode-invalidate-secctx s inode*

definition *security-inode-notifysecctx* :: *'s* ⇒ *inode* ⇒ *string* ⇒ *u32*⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-notifysecctx s inode ctx ctxlen* ≡ *hook-inode-notifysecctx s inode ctx ctxlen*

definition *security-inode-setsecctx* :: *'s* ⇒ *dentry* ⇒ *string* ⇒ *u32*⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-setsecctx s dentry ctx ctxlen* ≡ *hook-inode-setsecctx s dentry ctx ctxlen*

definition *security-inode-getsecctx* :: *'s* ⇒ *inode* ⇒ *string* ⇒ *u32*⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-getsecctx s dentry ctx ctxlen* ≡ *hook-inode-getsecctx s dentry ctx ctxlen*

definition *security-task-alloc* :: *'s* ⇒ *Task* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-alloc s task clone-flags* ≡ *hook-task-alloc s task clone-flags*

definition *security-task-free* :: *'s* ⇒ *Task* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-task-free s task* ≡ *hook-task-free s task*

definition *security-cred-alloc-blank* :: *'s* ⇒ *Cred* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
where *security-cred-alloc-blank s cred' gfp'* ≡ *hook-cred-alloc-blank s cred' gfp'*

definition *security-cred-free* :: *'s* ⇒ *Cred* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-cred-free s cred'* ≡ *hook-cred-free s cred'*

definition *security-prepare-creds* :: *'s* ⇒ *Cred* ⇒ *Cred* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
where *security-prepare-creds s new old gfp'* ≡ *hook-prepare-creds s new old gfp'*

definition *security-transfer-creds* :: *'s* ⇒ *Cred* ⇒ *Cred* ⇒ (*'s*, *unit*) *nondet-monad*

where *security-transfer-creds s new old* ≡ *hook-transfer-creds s new old*

definition *security-cred-getsecid* :: *'s* ⇒ *Cred* ⇒ *u32* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-cred-getsecid s c secid'* ≡ *do*
secid ← *return 0*;
hook-cred-getsecid s c secid
od

definition *security-task-fix-setuid* :: *'s* ⇒ *Cred* ⇒ *Cred* ⇒ *int* ⇒ (*'s*, *int*)

nondet-monad
where *security-task-fix-setuid s new old flgs* ≡ *hook-task-fix-setuid s new old flgs*

definition *security-task-setpgid* :: *'s* ⇒ *Task* ⇒ *pid-t* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-setpgid s p pgid* ≡ *hook-task-setpgid s p pgid*

definition *security-task-getpgid* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-getpgid s p* ≡ *hook-task-getpgid s p*

definition *security-task-getsid* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-getsid s p* ≡ *hook-task-getsid s p*

definition *security-task-getsecid* :: *'s* ⇒ *Task* ⇒ *u32* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-task-getsecid s c secid'* ≡ *do*
secid ← *return 0*;
hook-task-getsecid s c secid
od

definition *security-task-setnice* :: *'s* ⇒ *Task* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-setnice s p nice* ≡ *hook-task-setnice s p nice*

definition *security-task-setioprio* :: *'s* ⇒ *Task* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-setioprio s p ioprio* ≡ *hook-task-setioprio s p ioprio*

definition *security-task-getioprio* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-getioprio s p* ≡ *hook-task-getioprio s p*

definition *security-task-prlimit* :: *'s* ⇒ *Cred* ⇒ *Cred* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-prlimit s cred' tcred flgs* ≡ *hook-task-prlimit s cred' tcred flgs*

definition *security-task-setrlimit* :: *'s* ⇒ *Task* ⇒ *nat* ⇒ *rlimit* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-setrlimit s p res new-rlim* ≡ *hook-task-setrlimit s p res new-rlim*

definition *security-task-setscheduler* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
where*security-task-setscheduler s p* ≡ *hook-task-setscheduler s p*

definition *security-task-getscheduler* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-getscheduler s p* ≡ *hook-task-getscheduler s p*

definition *security-task-movememory* :: *'s* ⇒ *Task* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-movememory s p* ≡ *hook-task-movememory s p*

definition *security-task-kill* :: *'s* ⇒ *Task* ⇒ *siginfo* ⇒ *int* ⇒ *Cred option* ⇒ (*'s*, *int*) *nondet-monad*
where *security-task-kill s t info sig c* ≡ *hook-task-kill s t info sig c*

definition *security-task-prctl* :: *'s* ⇒ *int* ⇒ *nat* ⇒ *nat* ⇒*nat* ⇒ *nat* ⇒ (*'s*, *int*)

nondet-monad
where *security-task-prctl s opt arg2 arg3 arg4 arg5* $\equiv$ *do*
rc $\leftarrow$ *return*$(-ENOSYS)$;
thisrc $\leftarrow$*hook-task-prctl s opt arg2 arg3 arg4 arg5*;
if thisrc $\neq$ $(-ENOSYS)$
then
return thisrc
else
return rc
od

definition *security-task-to-inode* :: $'s \Rightarrow$ *Task* $\Rightarrow$ *inode* $\Rightarrow$ $('s,$ *unit*$)$ *nondet-monad*
where *security-task-to-inode s p inode* $\equiv$ *hook-task-to-inode s p inode*

definition *security-ipc-permission* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *nat* $\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
where *security-ipc-permission s ipcp flg* $\equiv$ *hook-ipc-permission s ipcp flg*

definition *security-ipc-getsecid* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *u32* $\Rightarrow$ $('s,$ *unit*$)$ *nondet-monad*
where *security-ipc-getsecid s ipcp secid'* $\equiv$ *do*
secid $\leftarrow$ *return 0*;
hook-ipc-getsecid s ipcp secid
od

definition *security-msg-msg-alloc* :: $'s \Rightarrow$ *msg-msg* $\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
where *security-msg-msg-alloc s msg* $\equiv$ *hook-msg-msg-alloc s msg*

definition *security-msg-msg-free* :: $'s \Rightarrow$ *msg-msg* $\Rightarrow$ $('s,$ *unit*$)$ *nondet-monad*
where *security-msg-msg-free s msg* $\equiv$ *hook-msg-msg-free s msg*

definition *security-msg-queue-alloc* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
where *security-msg-queue-alloc s msq* $\equiv$ *hook-msg-queue-alloc s msq*

definition *security-msg-queue-free* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ $('s,$ *unit*$)$ *nondet-monad*
where *security-msg-queue-free s msq* $\equiv$ *hook-msg-queue-free s msq*

definition *security-msg-queue-associate* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *int*$\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
where *security-msg-queue-associate s msq msqflg* $\equiv$ *hook-msg-queue-associate s msq msqflg*

definition *security-msg-queue-msgctl* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *IPC-CMD*$\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*
where *security-msg-queue-msgctl s msq cmd* $\equiv$ *hook-msg-queue-msgctl s msq cmd*

definition *security-msg-queue-msgsnd* :: $'s \Rightarrow$ *kern-ipc-perm* $\Rightarrow$ *msg-msg* $\Rightarrow$ *int*$\Rightarrow$ $('s,$ *int*$)$ *nondet-monad*

where *security-msg-queue-msgsnd s msq msg msqflg* ≡ *hook-msg-queue-msgsnd s msq msg msqflg*

definition *security-msg-queue-msgrcv* :: *'s ⇒ kern-ipc-perm ⇒ msg-msg⇒ Task ⇒ int ⇒ int⇒ ('s, int) nondet-monad*
where *security-msg-queue-msgrcv s msq msg target type m* ≡
hook-msg-queue-msgrcv s msq msg target type m

definition *security-shm-alloc* :: *'s ⇒ kern-ipc-perm ⇒ ('s, int) nondet-monad*
where *security-shm-alloc s shp* ≡ *hook-shm-alloc s shp*

definition *security-shm-free* :: *'s ⇒ kern-ipc-perm ⇒ ('s, unit) nondet-monad*
where *security-shm-free s shp* ≡ *hook-shm-free s shp*

definition *security-shm-associate* :: *'s ⇒ kern-ipc-perm ⇒ int⇒ ('s, int) nondet-monad*
where *security-shm-associate s shp shmflg* ≡ *hook-shm-associate s shp shmflg*

definition *security-shm-shmctl* :: *'s ⇒ kern-ipc-perm ⇒ IPC-CMD⇒ ('s, int) nondet-monad*
where *security-shm-shmctl s shp cmd* ≡ *hook-shm-shmctl s shp cmd*

definition *security-shm-shmat* :: *'s ⇒ kern-ipc-perm ⇒string ⇒int ⇒ ('s, int) nondet-monad*
where *security-shm-shmat s shp shmaddr shmflg* ≡ *hook-shm-shmat s shp shmaddr shmflg*

definition *security-sem-alloc* :: *'s ⇒ kern-ipc-perm ⇒ ('s, int) nondet-monad*
where *security-sem-alloc s sma* ≡ *hook-sem-alloc s sma*

definition *security-sem-free* :: *'s ⇒ kern-ipc-perm ⇒ ('s, unit) nondet-monad*
where *security-sem-free s sma* ≡ *hook-sem-free s sma*

definition *security-sem-associate*:: *'s ⇒ kern-ipc-perm ⇒ int⇒ ('s, int) nondet-monad*
where *security-sem-associate s sma semflg* ≡ *hook-sem-associate s sma semflg*

definition *security-sem-semctl* :: *'s ⇒ kern-ipc-perm ⇒ IPC-CMD⇒ ('s, int) nondet-monad*
where *security-sem-semctl s sma cmd* ≡ *hook-sem-semctl s sma cmd*

definition *security-sem-semop* :: *'s ⇒ kern-ipc-perm ⇒sembuf⇒nat ⇒ int⇒ ('s, int) nondet-monad*
where *security-sem-semop s sma sops nsops alter* ≡ *hook-sem-semop s sma sops nsops alter*

definition *file-inode* :: *Files ⇒ inode*
where *file-inode f* ≡ *f-inode f*

definition *fsnotify-perm*:: $'s \Rightarrow$ *Files* $\Rightarrow$ *mask* $\Rightarrow('s,\ int)$ *nondet-monad*
where *fsnotify-perm s file m* $\equiv$ *do*
path $\leftarrow$ *return*(*f-path file*);
inode $\leftarrow$ *return*(*file-inode file*);
return(*0*)
od

definition *security-file-permission* :: $'s \Rightarrow$ *Files* $\Rightarrow$ *int* $\Rightarrow ('s,\ int)$ *nondet-monad*
where *security-file-permission s file mask'* $\equiv$*do*
ret $\leftarrow$ *hook-file-permission s file mask'* ;
if ret $\neq$ *0 then return ret*
else fsnotify-perm s file mask'
od

definition *security-file-alloc* :: $'s \Rightarrow$ *Files* $\Rightarrow ('s,\ int)$ *nondet-monad*
where *security-file-alloc s file* $\equiv$ *hook-file-alloc s file*

definition *security-file-free* :: $'s \Rightarrow$ *Files* $\Rightarrow ('s,\ unit)$ *nondet-monad*
where *security-file-free s file* $\equiv$ *hook-file-free s file*

definition *security-file-ioctl* :: $'s \Rightarrow$ *Files* $\Rightarrow$ *IOC-DIR* $\Rightarrow$ *nat* $\Rightarrow ('s,\ int)$ *nondet-monad*
where *security-file-ioctl s file cmd arg* $\equiv$ *hook-file-ioctl s file cmd arg*

definition *mmap-capabilities* ::*Files* $\Rightarrow$ *nat*
where *mmap-capabilities f* $\equiv$ *0*

definition *mmap-prot-mmu* :: *Files* $\Rightarrow$ *nat* $\Rightarrow$ *nat*
where *mmap-prot-mmu f prot* $\equiv$
if CONFIG-MMU then
let
fop = *f-op f*;
caps = *mmap-capabilities f*
in
if fop = *fop-mmap-capabilities then*
if $\neg$((*caps AND NOMMU-MAP-EXEC*) $\neq$ *0*)
then prot
else
nat(*bitOR* (*int prot*) *PROT-EXEC*)
else
nat(*bitOR* (*int prot*) *PROT-EXEC*)
else
nat(*bitOR* (*int prot*) *PROT-EXEC*)

definition *mmap-prot* :: $'s \Rightarrow$ *Files option* $\Rightarrow$ *nat* $\Rightarrow ('s,\ int)$ *nondet-monad*
where *mmap-prot s file' prot* $\equiv$ *do*
flag1 $\leftarrow$ *return*(((*int prot*) *AND* (*bitOR PROT-READ PROT-EXEC*))
$\neq$ *PROT-READ*);

personality ← *return*(*personality* (*the*((*k-task s*)(*current s*))));
flag2 ← *return*((*personality AND READ-IMPLIES-EXEC*) = *0*);
rc ← (*if flag1* ∨ *flag2 then*
return (*int prot*)
else if file′ ≠ *None*
then return(*bitOR* (*int prot*) *PROT-EXEC*)
else if ¬(*path-noexec* (*f-path* (*the file′*))) *then*
return(*int* (*mmap-prot-mmu* (*the file′*) *prot*))
else
return((*int prot*))
);
return rc
od

definition *ima-file-mmap* :: *Files* ⇒ *nat* ⇒ *int*
where *ima-file-mmap file prot* ≡ *0*

definition *security-mmap-file* :: *′s* ⇒ *Files* ⇒ *nat* ⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*

where *security-mmap-file s file prot flgs* ≡ *do*
mprot ← *mmap-prot s* (*Some file*) *flgs*;
ret ← *hook-mmap-file s* (*Some file*) *prot* (*nat mprot*) *flgs*;
if ret ≠ *0 then*
return ret
else
return(*ima-file-mmap file prot*)
od

definition *security-mmap-addr* :: *′s* ⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-mmap-addr s addr* ≡ *hook-mmap-addr s addr*

definition *security-file-mprotect* :: *′s* ⇒ *vm-area-struct*⇒ *nat* ⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-mprotect s vma reqprot prot* ≡ *hook-file-mprotect s vma reqprot prot*

definition *security-file-lock* :: *′s* ⇒ *Files*⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-lock s file cmd* ≡ *hook-file-lock s file cmd*

definition *security-file-fcntl* :: *′s* ⇒ *Files*⇒ *nat* ⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-fcntl s file cmd arg* ≡ *hook-file-fcntl s file cmd arg*

definition *security-file-set-fowner* :: *′s* ⇒ *Files*⇒ (*′s*, *unit*) *nondet-monad*
where *security-file-set-fowner s file* ≡ *hook-file-set-fowner s file*

definition *security-file-send-sigiotask* :: *′s* ⇒ *Task*⇒ *fown-struct* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-send-sigiotask s tsk′ fown sig* ≡ *hook-file-send-sigiotask s tsk′ fown sig*

definition *security-file-receive* :: *'s* ⇒ *Files*⇒ (*'s*, *int*) *nondet-monad*
where *security-file-receive s file* ≡ *hook-file-receive s file*

definition *security-file-open* :: *'s* ⇒ *Files*⇒ (*'s*, *int*) *nondet-monad*
where *security-file-open s file* ≡ *do*
ret ← (*hook-file-open s file*);
rc ← (*if ret* ≠ *0 then*
return ret
else (*fsnotify-perm s file MAY-OPEN*));
return rc
od

definition *security-kernel-act-as* :: *'s* ⇒ *Cred* ⇒ *u32* ⇒ (*'s*, *int*) *nondet-monad*
where *security-kernel-act-as s new secid'* ≡ *hook-kernel-act-as s new secid'*

definition *security-kernel-create-files-as* :: *'s* ⇒ *Cred* ⇒ *inode* ⇒ (*'s*, *int*) *nondet-monad*
where *security-kernel-create-files-as s new inode* ≡ *hook-kernel-create-files-as s new inode*

definition *integrity-kernel-module-request* :: *'s* ⇒ *string* ⇒ (*'s*, *int*) *nondet-monad*
where *integrity-kernel-module-request s name* ≡ *return 0*

definition *security-kernel-module-request* :: *'s* ⇒ *string* ⇒ (*'s*, *int*) *nondet-monad*
where *security-kernel-module-request s name* ≡ *do*
ret ← *hook-kernel-module-request s name*;
if ret ≠ *0 then*
return ret
else
integrity-kernel-module-request s name
od

definition *ima-load-data* :: *'s* ⇒ *kernel-load-data-id* ⇒ (*'s*, *int*) *nondet-monad*
where *ima-load-data s kid* ≡ *return 0*

definition *security-kernel-load-data* :: *'s* ⇒ *kernel-load-data-id* ⇒ (*'s*, *int*) *nondet-monad*
where *security-kernel-load-data s kid* ≡ *do*
ret ← *hook-kernel-load-data s kid*;
if ret ≠ *0 then*
return ret
else
ima-load-data s kid
od

definition *ima-read-file* :: *'s* ⇒ *Files* ⇒*kernel-read-file-id* ⇒ (*'s*, *int*) *nondet-monad*
where *ima-read-file s file kid* ≡ *return 0*

definition *security-kernel-read-file* :: $'s \Rightarrow Files \Rightarrow kernel\text{-}read\text{-}file\text{-}id$
$\Rightarrow ('s,\ int)\ nondet\text{-}monad$
where *security-kernel-read-file s file kid* $\equiv$ *do*
ret $\leftarrow$ *hook-kernel-read-file s file kid*;
if ret $\neq$ *0 then*
return ret
else
ima-read-file s file kid
od

definition *ima-post-read-file* :: $'s \Rightarrow Files \Rightarrow string \Rightarrow nat \Rightarrow kernel\text{-}read\text{-}file\text{-}id$
$\Rightarrow ('s,\ int)\ nondet\text{-}monad$
where *ima-post-read-file s file buf size' kid* $\equiv$ *return 0*

definition *security-kernel-post-read-file* :: $'s \Rightarrow Files \Rightarrow string \Rightarrow nat \Rightarrow kernel\text{-}read\text{-}file\text{-}id$

$\Rightarrow ('s,\ int)\ nondet\text{-}monad$
where *security-kernel-post-read-file s file buf size' kid* $\equiv$ *do*
ret $\leftarrow$ *hook-kernel-post-read-file s file buf size' kid*;
if ret $\neq$ *0 then*
return ret
else
ima-post-read-file s file buf size' kid
od

definition *security-dentry-init-security* :: $'s \Rightarrow dentry \Rightarrow mode \Rightarrow string \Rightarrow string$
$\Rightarrow int$
$\Rightarrow ('s,\ int)\ nondet\text{-}monad$
where *security-dentry-init-security s dentry m name ctx xtxlen* $\equiv$
hook-dentry-init-security s dentry m name ctx xtxlen

definition *security-dentry-create-files-as* :: $'s \Rightarrow dentry \Rightarrow mode \Rightarrow string \Rightarrow Cred$
$\Rightarrow Cred$
$\Rightarrow ('s,\ int)\ nondet\text{-}monad$
where *security-dentry-create-files-as s dentry m name old new* $\equiv$
hook-dentry-create-files-as s dentry m name old new

definition *security-d-instantiate*:: $'s \Rightarrow dentry \Rightarrow inode \Rightarrow ('s,\ unit)\ nondet\text{-}monad$
where *security-d-instantiate s dentry inode* $\equiv$
if unlikely (*IS-PRIVATE* (*inode*)) *then*
return ()
else
hook-d-instantiate s dentry (*Some inode*)

definition *security-getprocattr* :: $'s \Rightarrow Task \Rightarrow string \Rightarrow string \Rightarrow ('s,\ int)$
nondet-monad
where *security-getprocattr s p name value* $\equiv$ *hook-getprocattr s p name value*

definition *security-setprocattr* :: $'s \Rightarrow$ *string* $\Rightarrow$ *string* $\Rightarrow$ *int* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-setprocattr s name value size'* $\equiv$ *hook-setprocattr s name value size'*

definition *security-netlink-send* :: $'s \Rightarrow$ *sock* $\Rightarrow$ *sk-buff* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-netlink-send s sk' skb* $\equiv$ *hook-netlink-send s sk' skb*

definition *security-ismaclabel* :: $'s \Rightarrow$ *xattr* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-ismaclabel s name* $\equiv$ *hook-ismaclabel s name*

definition *security-secid-to-secctx* :: $'s \Rightarrow$ *u32* $\Rightarrow$ *string* $\Rightarrow$ *u32* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-secid-to-secctx s secid' secdata seclen* $\equiv$
hook-secid-to-secctx s secid' secdata seclen

definition *security-secctx-to-secid* :: $'s \Rightarrow$ *string* $\Rightarrow$ *u32* $\Rightarrow$ *u32* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-secctx-to-secid s secdata seclen secid'* $\equiv$ *do*
secid $\leftarrow$ *return 0*;
hook-secctx-to-secid s secdata seclen secid
od

definition *security-release-secctx* :: $'s \Rightarrow$ *string* $\Rightarrow$ *u32* $\Rightarrow$ $('s, unit)$ *nondet-monad*
where *security-release-secctx s secdata seclen* $\equiv$ *hook-release-secctx s secdata seclen*

definition *security-unix-stream-connect* :: $'s \Rightarrow$ *sock* $\Rightarrow$ *sock* $\Rightarrow$ *sock* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-unix-stream-connect s sock other newsk* $\equiv$ *hook-unix-stream-connect s sock other newsk*

definition *security-unix-may-send* :: $'s \Rightarrow$ *socket* $\Rightarrow$ *socket* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-unix-may-send s sock other* $\equiv$ *hook-unix-may-send s sock other*

definition *security-socket-create* :: $'s \Rightarrow$ *Sk-Family* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-socket-create s family type pro kern* $\equiv$ *hook-socket-create s family type pro kern*

definition *security-socket-post-create* :: $'s \Rightarrow$ *socket* $\Rightarrow$ *Sk-Family* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int*
$\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-socket-post-create s sock family type pro kern* $\equiv$
hook-socket-post-create s sock family type pro kern

definition *security-socket-socketpair* :: $'s \Rightarrow$ *socket* $\Rightarrow$ *socket* $\Rightarrow$ $('s, int)$ *nondet-monad*
where *security-socket-socketpair s socka sockb* $\equiv$ *hook-socket-socketpair s socka sockb*

definition *security-socket-bind* :: $'s \Rightarrow$ *socket* $\Rightarrow$ *sockaddr* $\Rightarrow$ *int* $\Rightarrow$ $('s, int)$

nondet-monad
  where *security-socket-bind s sock address addrlen* ≡ *hook-socket-bind s sock address addrlen*

definition *security-socket-connect* :: *'s* ⇒ *socket* ⇒ *sockaddr* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-connect s sock address addrlen* ≡ *hook-socket-connect s sock address addrlen*

definition *security-socket-listen* :: *'s* ⇒ *socket* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-listen s sock backlog* ≡ *hook-socket-listen s sock backlog*

definition *security-socket-accept* :: *'s* ⇒ *socket* ⇒ *socket* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-accept s sock newsock* ≡ *hook-socket-accept s sock newsock*

definition *security-socket-sendmsg* :: *'s* ⇒ *socket* ⇒ *msghdr* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-sendmsg s sock msg size'* ≡ *hook-socket-sendmsg s sock msg size'*

definition *security-socket-recvmsg* :: *'s* ⇒ *socket* ⇒ *msghdr* ⇒ *int* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-recvmsg s sock msg size' flgs* ≡ *hook-socket-recvmsg s sock msg size' flgs*

definition *security-socket-getsockname* :: *'s* ⇒ *socket* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-getsockname s sock* ≡ *hook-socket-getsockname s sock*

definition *security-socket-getpeername* :: *'s* ⇒ *socket* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-getpeername s sock* ≡ *hook-socket-getpeername s sock*

definition *security-socket-getsockopt* :: *'s* ⇒ *socket* ⇒*int* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-getsockopt s sock level' optname* ≡
    hook-socket-getsockopt s sock level' optname

definition *security-socket-setsockopt* :: *'s* ⇒ *socket* ⇒*int* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-setsockopt s sock level' optname* ≡
    hook-socket-setsockopt s sock level' optname

definition *security-socket-shutdown*:: *'s* ⇒ *socket* ⇒*int* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-socket-shutdown s sock how* ≡ *hook-socket-shutdown s sock how*

definition *security-sock-rcv-skb* :: *'s* ⇒ *sock* ⇒*sk-buff* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-sock-rcv-skb s sock skb* ≡ *hook-sock-rcv-skb s sock skb*

definition *security-socket-getpeersec-stream* :: *'s* ⇒ *socket* ⇒ *string* ⇒*int* ⇒ *nat*

$\Rightarrow$ ('s, int) nondet-monad
where *security-socket-getpeersec-stream s sock optval optlen len'* ≡
hook-socket-getpeersec-stream s sock optval optlen len'

definition *security-socket-getpeersec-dgram* :: 's ⇒ socket ⇒ sk-buff option⇒ u32

⇒ ('s, int) nondet-monad
where *security-socket-getpeersec-dgram s sock skb secid'* ≡
hook-socket-getpeersec-dgram s sock skb secid'

definition *security-sk-alloc* :: 's ⇒ sock ⇒ int ⇒ gfp-t ⇒ ('s, int) nondet-monad
where *security-sk-alloc s sk' family priority* ≡ *hook-sk-alloc s sk' family priority*

definition *security-sk-free* :: 's ⇒ sock ⇒ ('s, unit) nondet-monad
where *security-sk-free s sock* ≡ *hook-sk-free s sock*

definition *security-sk-clone* :: 's ⇒ sock ⇒ sock ⇒ ('s, unit) nondet-monad
where *security-sk-clone s sk' newsk* ≡ *hook-sk-clone s sk' newsk*

definition *security-sk-classify-flow* :: 's ⇒ sock ⇒ flowi ⇒ ('s, unit) nondet-monad
where *security-sk-classify-flow s sock' fl* ≡ *hook-sk-classify-flow s sock' fl*

definition *security-req-classify-flow* :: 's ⇒ request-sock⇒ flowi ⇒ ('s, unit) nondet-monad
where *security-req-classify-flow s req fl* ≡ *hook-req-classify-flow s req fl*

definition *security-sock-graft* :: 's ⇒ sock ⇒ socket ⇒ ('s, unit) nondet-monad
where *security-sock-graft s sk' parent'* ≡ *hook-sock-graft s sk' parent'*

definition *security-inet-conn-request* :: 's ⇒ sock ⇒ sk-buff ⇒request-sock
⇒ ('s, int) nondet-monad
where *security-inet-conn-request s sk' skb req* ≡ *hook-inet-conn-request s sk' skb req*

definition *security-inet-csk-clone* :: 's ⇒ sock ⇒ request-sock ⇒ ('s, unit) nondet-monad
where *security-inet-csk-clone s newsk req* ≡ *hook-inet-csk-clone s newsk req*

definition *security-inet-conn-established* :: 's ⇒ sock ⇒ sk-buff ⇒ ('s, unit) nondet-monad
where *security-inet-conn-established s sk' skb* ≡ *hook-inet-conn-established s sk' skb*

definition *security-secmark-relabel-packet* :: 's ⇒ u32 ⇒ ('s, int) nondet-monad
where *security-secmark-relabel-packet s secid'* ≡ *hook-secmark-relabel-packet s secid'*

definition *security-secmark-refcount-inc* :: 's ⇒ ('s, unit) nondet-monad
where *security-secmark-refcount-inc s* ≡ *hook-secmark-refcount-inc s*

definition *security-secmark-refcount-dec* :: $'s \Rightarrow ('s,\ unit)$ *nondet-monad*
where*security-secmark-refcount-dec s* $\equiv$ *hook-secmark-refcount-dec s*

definition *security-tun-dev-alloc-security* :: $'s \Rightarrow 'h \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-tun-dev-alloc-security s security* $\equiv$ *hook-tun-dev-alloc-security s security*

definition *security-tun-dev-free-security* :: $'s \Rightarrow 'h \Rightarrow ('s,\ unit)$ *nondet-monad*
where *security-tun-dev-free-security s security* $\equiv$ *hook-tun-dev-free-security s security*

definition *security-tun-dev-create* :: $'s \Rightarrow ('s,\ int)$ *nondet-monad*
where*security-tun-dev-create s* $\equiv$ *hook-tun-dev-create s*

definition *security-tun-dev-attach-queue* :: $'s \Rightarrow 'h \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-tun-dev-attach-queue s security* $\equiv$ *hook-tun-dev-attach-queue s security*

definition *security-tun-dev-attach* :: $'s \Rightarrow sock \Rightarrow 'h \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-tun-dev-attach s sk' security* $\equiv$ *hook-tun-dev-attach s sk' security*

definition *security-tun-dev-open* :: $'s \Rightarrow 'h \Rightarrow ('s,\ int)$ *nondet-monad*
where*security-tun-dev-open s security* $\equiv$ *hook-tun-dev-open s security*

definition *security-sctp-assoc-request* :: $'s \Rightarrow sctp\text{-}endpoint \Rightarrow sk\text{-}buff \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-sctp-assoc-request s ep skb* $\equiv$ *hook-sctp-assoc-request s ep skb*

definition *security-sctp-bind-connect* :: $'s \Rightarrow sock \Rightarrow int \Rightarrow sockaddr \Rightarrow int \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-sctp-bind-connect s sk' optname address addrlen* $\equiv$
hook-sctp-bind-connect s sk' optname address addrlen

definition *security-sctp-sk-clone* :: $'s \Rightarrow sctp\text{-}endpoint \Rightarrow sock \Rightarrow sock \Rightarrow ('s,\ unit)$ *nondet-monad*
where *security-sctp-sk-clone s ep sk' newsk* $\equiv$ *hook-sctp-sk-clone s ep sk' newsk*

definition *security-ib-pkey-access* :: $'s \Rightarrow 'i \Rightarrow nat \Rightarrow nat \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-ib-pkey-access s sec subnet-prefix pkey* $\equiv$
hook-ib-pkey-access s sec subnet-prefix pkey

definition *security-ib-endport-manage-subnet*:: $'s \Rightarrow 'i \Rightarrow string \Rightarrow nat \Rightarrow ('s,\ int)$ *nondet-monad*
where *security-ib-endport-manage-subnet s sec dev-name port-num* $\equiv$
hook-ib-endport-manage-subnet s sec dev-name port-num

definition *security-ib-alloc-security* :: $'s \Rightarrow 'i\ list \Rightarrow ('s,\ int)$ *nondet-monad*

where *security-ib-alloc-security s sec* ≡ *hook-ib-alloc-security s sec*

definition *security-ib-free-security* :: *'s* ⇒ *'i list* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-ib-free-security s sec* ≡ *hook-ib-free-security s sec*

definition *security-path-unlink* :: *'s* ⇒ *path* ⇒ *dentry* ⇒(*'s*, *int*) *nondet-monad*
where *security-path-unlink s dir dentry* ≡
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*p-dentry dir*))))
then
return 0
else
hook-path-unlink s dir dentry

definition *security-path-mkdir* :: *'s* ⇒ *path* ⇒ *dentry* ⇒*nat* ⇒(*'s*, *int*) *nondet-monad*
where *security-path-mkdir s dir dentry m* ≡
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*p-dentry dir*))))
then
return 0
else
hook-path-mkdir s dir dentry m

definition *security-path-rmdir* :: *'s* ⇒ *path* ⇒ *dentry* ⇒(*'s*, *int*) *nondet-monad*
where *security-path-rmdir s dir dentry* ≡
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*p-dentry dir*))))
then
return 0
else
hook-path-rmdir s dir dentry

definition *security-path-mknod* :: *'s* ⇒ *path* ⇒ *dentry* ⇒*nat* ⇒ *nat*⇒(*'s*, *int*) *nondet-monad*
where *security-path-mknod s dir dentry m dev*≡
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*p-dentry dir*))))
then return 0
else hook-path-mknod s dir dentry m dev

definition *security-path-truncate* :: *'s* ⇒ *path* ⇒(*'s*, *int*) *nondet-monad*
where *security-path-truncate s dir* ≡
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*p-dentry dir*))))
then
return 0
else
hook-path-truncate s dir

definition *security-path-symlink* :: *'s* ⇒ *path* ⇒ *dentry* ⇒ *string* ⇒(*'s*, *int*) *nondet-monad*
where *security-path-symlink s dir dentry old-name* ≡
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*p-dentry dir*))))

then
return 0
else
hook-path-symlink s dir dentry old-name

definition *security-path-link* :: $'s \Rightarrow$ *dentry* $\Rightarrow$*path* $\Rightarrow$ *dentry* $\Rightarrow$($'s$, *int*) *nondet-monad*
where *security-path-link s old-dentry new-dir new-dentry* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*old-dentry*))))
then
return 0
else
hook-path-link s old-dentry new-dir new-dentry

definition *security-path-rename* :: $'s \Rightarrow$ *path* $\Rightarrow$ *dentry* $\Rightarrow$ *path* $\Rightarrow$ *dentry* $\Rightarrow$ *nat*

$\Rightarrow$ ($'s$, *int*) *nondet-monad*
where *security-path-rename s old-dir old-dentry new-dir new-dentry flgs* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s old-dentry*))) $\vee$
((*d-is-positive new-dentry*) $\wedge$ *IS-PRIVATE* (*the*(*d-backing-inode s*
new-dentry)) $\neq$ *0*)
then
return 0
else if (((*int flgs*) *AND RENAME-EXCHANGE*) $\neq$ *0*) *then*
do
err $\leftarrow$ (*hook-path-rename s new-dir new-dentry old-dir old-dentry*);
if err $\neq$ *0 then*
return err
else
(*hook-path-rename s old-dir old-dentry new-dir new-dentry*)
od
else
(*hook-path-rename s old-dir old-dentry new-dir new-dentry*)

definition *security-path-chmod* :: $'s \Rightarrow$ *path* $\Rightarrow$ *nat*$\Rightarrow$($'s$, *int*) *nondet-monad*
where *security-path-chmod s path m* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*p-dentry path*))))
then
return 0
else
hook-path-chmod s path m

definition *security-path-chown* :: $'s \Rightarrow$ *path* $\Rightarrow$ *kuid-t* $\Rightarrow$ *kgid-t* $\Rightarrow$($'s$, *int*) *nondet-monad*
where *security-path-chown s path uid' gid'* $\equiv$
if unlikely (*IS-PRIVATE* (*the*(*d-backing-inode s* (*p-dentry path*))))
then return 0
else hook-path-chown s path uid' gid'

definition *security-path-chroot* :: $'s \Rightarrow$ *path* $\Rightarrow$($'s$, *int*) *nondet-monad*
where *security-path-chroot s path* $\equiv$ *hook-path-chroot s path*

definition *security-key-alloc* :: *'s* ⇒ *key* ⇒ *Cred* ⇒ *nat* ⇒(*'s*, *int*) *nondet-monad*
where *security-key-alloc s key' c flgs* ≡ *hook-key-alloc s key' c flgs*

definition *security-key-free* :: *'s* ⇒ *key* ⇒(*'s*, *unit*) *nondet-monad*
where *security-key-free s key'* ≡ *hook-key-free s key'*

definition *security-key-permission* :: *'s* ⇒ *key-ref-t* ⇒ *Cred* ⇒ *nat* ⇒(*'s*, *int*) *nondet-monad*
where *security-key-permission s key-ref c perm* ≡ *hook-key-permission s key-ref c perm*

definition *security-key-getsecurity* :: *'s* ⇒ *key* ⇒ *string* ⇒(*'s*, *int*) *nondet-monad*
where *security-key-getsecurity s key' buffer* ≡ *hook-key-getsecurity s key' buffer*

definition *security-audit-rule-init* :: *'s* ⇒ *nat* ⇒ *enum-audit* ⇒ *string* ⇒ *string*
⇒ (*'s*, *int*) *nondet-monad*
where *security-audit-rule-init s field op rulestr lsmrule* ≡
hook-audit-rule-init s field op rulestr lsmrule

definition *security-audit-rule-known* :: *'s* ⇒ *audit-krule* ⇒ (*'s*, *int*) *nondet-monad*
where *security-audit-rule-known s krule* ≡ *hook-audit-rule-known s krule*

definition *security-audit-rule-match* :: *'s* ⇒ *nat* ⇒ *nat* ⇒ *enum-audit* ⇒ *string* ⇒ *audit-context*
⇒ (*'s*, *int*) *nondet-monad*
where *security-audit-rule-match s secid' field op lsmrule actx* ≡
hook-audit-rule-match s secid' field op lsmrule actx

definition *security-audit-rule-free* :: *'s* ⇒ *string* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-audit-rule-free s lsmrule* ≡ *hook-audit-rule-free s lsmrule*

definition *security-xfrm-policy-alloc* :: *'s* ⇒ *xfrm-sec-ctx* ⇒ *xfrm-user-sec-ctx* ⇒ *gfp-t*
⇒ (*'s*, *int*) *nondet-monad*
where *security-xfrm-policy-alloc s ctxp sec-ctx gfp'* ≡ *hook-xfrm-policy-alloc s ctxp sec-ctx gfp'*

definition *security-xfrm-policy-clone* :: *'s* ⇒ *xfrm-sec-ctx* ⇒ *xfrm-user-sec-ctx*
⇒ (*'s*, *int*) *nondet-monad*
where *security-xfrm-policy-clone s old-ctx new-ctxp* ≡ *hook-xfrm-policy-clone s old-ctx new-ctxp*

definition *security-xfrm-policy-free* :: *'s* ⇒ *xfrm-sec-ctx* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-xfrm-policy-free s ctx* ≡ *hook-xfrm-policy-free s ctx*

definition *security-xfrm-policy-delete* :: *'s* ⇒ *xfrm-sec-ctx* ⇒ (*'s*, *int*) *nondet-monad*
where *security-xfrm-policy-delete s ctx* ≡ *hook-xfrm-policy-delete s ctx*

definition *security-xfrm-state-alloc* :: *'s* ⇒ *xfrm-state* ⇒ *xfrm-sec-ctx* ⇒ (*'s*, *int*) *nondet-monad*
where *security-xfrm-state-alloc s x sec-ctx* ≡ *hook-xfrm-state-alloc s x sec-ctx*

definition *security-xfrm-state-alloc-acquire* :: *'s* ⇒ *xfrm-state* ⇒ *xfrm-sec-ctx* ⇒ *u32*
⇒ (*'s*, *int*) *nondet-monad*
where *security-xfrm-state-alloc-acquire s x plosec secid'* ≡
hook-xfrm-state-alloc-acquire s x plosec secid'

definition *security-xfrm-state-delete* :: *'s* ⇒ *xfrm-state* ⇒(*'s*, *int*) *nondet-monad*
where *security-xfrm-state-delete s x* ≡ *hook-xfrm-state-delete s x*

definition *security-xfrm-state-free* :: *'s* ⇒ *xfrm-state* ⇒(*'s*, *unit*) *nondet-monad*
where *security-xfrm-state-free s x* ≡ *hook-xfrm-state-free s x*

definition *security-xfrm-policy-lookup* :: *'s* ⇒ *xfrm-sec-ctx* ⇒ *u32* ⇒ *u8* ⇒(*'s*, *int*) *nondet-monad*
where *security-xfrm-policy-lookup s ctx fl-secid dir* ≡ *hook-xfrm-policy-lookup s ctx fl-secid dir*

definition *security-xfrm-state-pol-flow-match* :: *'s* ⇒ *xfrm-state* ⇒ *xfrm-policy* ⇒ *flowi*
⇒ (*'s*, *int*) *nondet-monad*
where *security-xfrm-state-pol-flow-match s x xp fl*≡ *return 1*

definition *security-xfrm-decode-session* :: *'s* ⇒ *sk-buff* ⇒ *u32* ⇒(*'s*, *int*) *nondet-monad*
where *security-xfrm-decode-session s skb secid'* ≡ *hook-xfrm-decode-session s skb 1*

definition *security-skb-classify-flow* :: *'s* ⇒ *sk-buff* ⇒ *flowi* ⇒(*'s*, *unit*) *nondet-monad*
where *security-skb-classify-flow s skb fl* ≡ *hook-skb-classify-flow s skb fl*

definition *security-bpf* :: *'s* ⇒ *int* ⇒ *bpf-attr* ⇒ *nat* ⇒(*'s*, *int*) *nondet-monad*
where *security-bpf s cmd attr' size'* ≡ *hook-bpf s cmd attr' size'*

definition *security-bpf-map* :: *'s* ⇒ *bpf-map* ⇒ *mode* ⇒(*'s*, *int*) *nondet-monad*
where *security-bpf-map s bmap fmode* ≡ *hook-bpf-map s bmap fmode*

definition *security-bpf-prog* :: *'s* ⇒ *bpf-prog* ⇒(*'s*, *int*) *nondet-monad*
where *security-bpf-prog s prog* ≡ *hook-bpf-prog s prog*

definition *security-bpf-map-alloc* :: $'s \Rightarrow$ *bpf-map* $\Rightarrow('s,$ *int*$)$ *nondet-monad*
where *security-bpf-map-alloc s bmap* $\equiv$ *hook-bpf-map-alloc s bmap*

definition *security-bpf-map-free* :: $'s \Rightarrow$ *bpf-map* $\Rightarrow('s,$ *unit*$)$ *nondet-monad*
where *security-bpf-map-free s bmap* $\equiv$ *hook-bpf-map-free s bmap*

definition *security-bpf-prog-alloc* :: $'s \Rightarrow$ *bpf-prog-aux* $\Rightarrow('s,$ *int*$)$ *nondet-monad*
where *security-bpf-prog-alloc s aux'* $\equiv$ *hook-bpf-prog-alloc s aux'*

definition *security-bpf-prog-free* :: $'s \Rightarrow$ *bpf-prog-aux* $\Rightarrow('s,$ *unit*$)$ *nondet-monad*
where *security-bpf-prog-free s aux'* $\equiv$ *hook-bpf-prog-free s aux'*

27.3 func lemma

27.4 binder state lemma

lemma *security-binder-set-context-mgr-notchgstate* :
$\bigwedge sa$. $\{|\lambda s . s = sa|\}$ *security-binder-set-context-mgr sa mgr* $\{|\lambda r\ s.\ s = sa|\}$
using *security-binder-set-context-mgr-def stb-binder-set-context-mgr*
by *simp*

lemma *security-binder-transaction-notchgstate* :
$\bigwedge sa$. $\{|\lambda s . s = sa|\}$ *security-binder-transaction sa from to* $\{|\lambda r\ s.\ s = sa|\}$
using *security-binder-transaction-def stb-binder-transaction*
by *simp*

lemma *security-binder-transfer-binder-notchgstate* :
$\bigwedge sa$. $\{|\lambda s . s = sa|\}$ *security-binder-transfer-binder s from to* $\{|\lambda r\ s.\ s = sa|\}$
using *security-binder-transfer-binder-def stb-binder-transfer-binder*
by *simp*

lemma *security-binder-transfer-file-notchgstate* :
$\bigwedge sa$. $\{|\lambda s . s = sa|\}$ *security-binder-transfer-file s from to file* $\{|\lambda r\ s.\ s = sa|\}$
using *security-binder-transfer-file-def stb-binder-transfer-file*
by *simp*

27.5 ptrace state lemma

lemma *security-ptrace-access-check-notchgstate* :
$\bigwedge sa$. $\{|\lambda s . s = sa|\}$ *security-ptrace-access-check s child m* $\{|\lambda r\ s.\ s = sa|\}$
using *security-ptrace-access-check-def stb-ptrace-access-check*
by *simp*

lemma *security-ptrace-traceme-notchgstate* :
$\bigwedge sa$. $\{|\lambda s . s = sa|\}$ *security-ptrace-traceme s parent'* $\{|\lambda r\ s.\ s = sa|\}$
using *security-ptrace-traceme-def stb-ptrace-traceme*
by *simp*

27.6 file state lemma

lemma *security-file-permission-notchgstate* :
 $\bigwedge sa$. $\{\lambda s . s = sa\}$ *security-file-permission sa file mask'* $\{\lambda r\ s.\ s = sa\}$
 unfolding *security-file-permission-def fsnotify-perm-def*
 apply (*simp add*: *bind-def return-def split-def valid-def*)
 apply *wpsimp*
 using *stb-file-permission*
 apply *auto*
 apply (*simp add*: *valid-def split-def*)
 by *fastforce*

lemma *security-file-ioctl-notchgstate* :
 $\bigwedge sa$ *file*. $\{\lambda s . s = sa\}$ *security-file-ioctl sa file cmd arg* $\{\lambda r\ s.\ s = sa\}$
 unfolding *security-file-ioctl-def*
 using *stb-file-ioctl*
 apply *auto*[*1*]
 done

lemma *security-mmap-addr-notchgstate* :
 $\bigwedge sa$. $\{\lambda s . s = sa\}$ *security-mmap-addr sa addr* $\{\lambda r\ s.\ s = sa\}$
 unfolding *security-mmap-addr-def*
 using *stb-mmap-addr*
 by *simp*

lemma *security-mmap-file-notchgstate* :
 $\bigwedge sa$. $\{\lambda s . s = sa\}$ *security-mmap-file sa file prot flgs* $\{\lambda r\ s.\ s = sa\}$
 unfolding *security-mmap-file-def bind-def ima-file-mmap-def*
 apply *wpsimp*
 apply(*simp add*: *valid-def mmap-prot-def return-def bind-def*)
 apply(*simp add*: *PROT-READ-def PROT-EXEC-def return-def READ-IMPLIES-EXEC-def*)

 using *stb-mmap-file*
 apply (*simp add*: *valid-def split-def*)
 apply *auto*[*1*]
 by(*simp add*: *return-def*) +

lemma *security-file-mprotect-notchgstate* :
 $\bigwedge sa$. $\{\lambda s . s = sa\}$ *security-file-mprotect sa vma reqprot prot* $\{\lambda r\ s.\ s = sa\}$
 unfolding *security-file-mprotect-def*
 using *stb-file-mprotect*
 by *simp*

lemma *security-file-lock-notchgstate* :
 $\bigwedge sa$. $\{\lambda s . s = sa\}$ *security-file-lock sa file cmd* $\{\lambda r\ s.\ s = sa\}$
 unfolding *security-file-lock-def*
 using *stb-file-lock*
 by *simp*

lemma *security-file-fcntl-notchgstate* :

$\bigwedge sa$. $\{\!\lvert \lambda s . s = sa \rvert\!\}$ *security-file-fcntl sa file cmd arg* $\{\!\lvert \lambda r\ s.\ s = sa \rvert\!\}$
unfolding *security-file-fcntl-def*
using *stb-file-fcntl*
by *simp*

lemma *security-file-send-sigiotask-notchgstate* :
$\bigwedge sa$. $\{\!\lvert \lambda s . s = sa \rvert\!\}$ *security-file-send-sigiotask sa tsk' fown sig* $\{\!\lvert \lambda r\ s.\ s = sa \rvert\!\}$
unfolding *security-file-send-sigiotask-def*
using *stb-file-send-sigiotask*
by *simp*

lemma *security-file-receive-notchgstate* :
$\bigwedge sa$. $\{\!\lvert \lambda s . s = sa \rvert\!\}$ *security-file-receive sa file* $\{\!\lvert \lambda r\ s.\ s = sa \rvert\!\}$
unfolding *security-file-receive-def*
using *stb-file-receive*
by *simp*

lemma *security-file-open-notchgstate* :
$\bigwedge sa$. $\{\!\lvert \lambda s . s = sa \rvert\!\}$ *security-file-open sa file* $\{\!\lvert \lambda r\ s.\ s = sa \rvert\!\}$
unfolding *security-file-open-def return-def bind-def valid-def*
apply *auto*
using *stb-file-open*
apply(*simp add*: *valid-def*)
apply *simp*
using *fsnotify-perm-def* **apply** *auto*
by (*smt case-prodD fst-conv return-def singleton-iff*)

lemma *do-ioctl-state*:
$\bigwedge sa$. $\{\!\lvert \lambda s . s = sa \rvert\!\}$ *security-file-ioctl sa file cmd arg* $\{\!\lvert \lambda r\ s.\ s = sa \rvert\!\}$ $\wedge$
det (*security-file-ioctl sa file cmd arg*)$\longrightarrow$
snd (*the-run-state* (*security-file-ioctl sa file cmd arg*) *sa*)= *sa*
apply(*simp add*: *valid-def*)
apply *auto*[*1*]
using *all-not-in-conv det-def fst-conv insert-not-empty*
the-run-state-def the-run-state-det prod.case-eq-if
by *smt*

27.7 cap state lemma

lemma *security-capget-notchgstate* :
$\bigwedge sa$. $\{\!\lvert \lambda s . s = sa \rvert\!\}$*security-capget s target effective inheritable permitted* $\{\!\lvert \lambda r\ s.\ s = sa \rvert\!\}$
unfolding *security-capget-def*
using *stb-capget*
by *simp*

lemma *security-capset-notchgstate* :
$\bigwedge sa$. $\{\!\lvert \lambda s . s = sa \rvert\!\}$*security-capset s new old effective inheritable permitted*
$\{\!\lvert \lambda r\ s.\ s = sa \rvert\!\}$

unfolding *security-capset-def*
using *stb-capset*
by *simp*

lemma *security-capable-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s . s = sa|\!\}$ *security-capable s c ns cap*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-capable-def*
using *stb-capable*
by *simp*

lemma *security-capable-noaudit-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s . s = sa|\!\}$ *security-capable-noaudit s c ns cap*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-capable-noaudit-def*
using *stb-capable-noaudit*
by *simp*

lemma *security-quotactl-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s . s = sa|\!\}$ *security-quotactl s cmds t id′ sb*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-quotactl-def*
using *stb-quotactl*
by *simp*

lemma *security-quota-on-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s . s = sa|\!\}$ *security-quota-on s dentry*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-quota-on-def*
using *stb-quota-on*
by *simp*

lemma *security-settime64-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s . s = sa|\!\}$ *security-settime64 s ts tz*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-settime64-def*
using *stb-settime64*
by *simp*

lemma *security-vm-enough-memory-mm-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s . s = sa|\!\}$ *security-vm-enough-memory-mm s mm′ pages*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-vm-enough-memory-mm-def bind-def*
apply *auto*
using *stb-vm-enough-memory-mm*
by(*simp add*: *valid-def return-def split-def*)

lemma *security-syslog-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s . s = sa|\!\}$ *security-syslog s type*

$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-syslog-def*
using *stb-syslog*
by *simp*

lemma *security-bprm-set-creds-notchgstate* :
$\bigwedge sa\ .\ \{\!|\lambda s\ .\ s = sa|\!\}$ *security-bprm-set-creds s bprm*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-bprm-set-creds-def*
using *stb-bprm-set-creds*
by *simp*

lemma *security-bprm-check-notchgstate* :
$\bigwedge sa\ .\ \{\!|\lambda s\ .\ s = sa|\!\}$ *security-bprm-check s bprm*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-bprm-check-def*
using *stb-bprm-check*
by(*simp add*: *valid-def return-def bind-def split-def*)

lemma *security-bprm-committing-creds-notchgstate* :
$\bigwedge sa\ .\ \{\!|\lambda s\ .\ s = sa|\!\}$ *security-bprm-committing-creds s bprm*
$\{\!|\lambda r\ s.\ True|\!\}$
unfolding *security-bprm-committing-creds-def*
using *stb-bprm-committing-creds*
by *simp*

lemma *security-bprm-committed-creds-notchgstate* :
$\bigwedge sa\ .\ \{\!|\lambda s\ .\ s = sa|\!\}$ *security-bprm-committed-creds s bprm*
$\{\!|\lambda r\ s.\ True|\!\}$
unfolding *security-bprm-committed-creds-def*
using *stb-bprm-committed-creds*
by *simp*

27.8 sb state lemma

lemma *security-sb-alloc-notchgstate* :
$\bigwedge sa\ .\ \{\!|\lambda s\ .\ s = sa|\!\}$*security-sb-alloc s sb*
$\{\!|\lambda r\ s.\ r = 0 \lor r = -ENOMEM|\!\}$
unfolding *security-sb-alloc-def*
using *stb-sb-alloc-hook*
by *simp*

lemma *security-sb-free-r* :
$\bigwedge sa\ .\ \{\!|\lambda s\ .\ s = sa|\!\}$*security-sb-free s sb*
$\{\!|\lambda r\ s.\ r = unit|\!\}$
unfolding *security-sb-free-def*
using *stb-sb-free*
by *simp*

lemma *security-sb-copy-data-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s\ .\ s = sa|\!\}$ *security-sb-copy-data s orig copy*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-sb-copy-data-def*
using *stb-sb-copy-data*
by(*simp add*: *valid-def return-def bind-def split-def*)

lemma *security-sb-remount-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s\ .\ s = sa|\!\}$ *security-sb-remount s sb data*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-sb-remount-def*
using *stb-sb-remount*
by(*simp add*: *valid-def*)

lemma *security-sb-kern-mount-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s\ .\ s = sa|\!\}$ *security-sb-kern-mount s sb flgs data*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-sb-kern-mount-def*
using *stb-sb-kern-mount*
by(*simp add*: *valid-def*)

lemma *security-sb-show-options-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s\ .\ s = sa|\!\}$*security-sb-show-options s m sb*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-sb-show-options-def*
using *stb-sb-show-options*
by(*simp add*: *valid-def*)

lemma *security-sb-statfs-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s\ .\ s = sa|\!\}$*security-sb-statfs s dentry*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-sb-statfs-def*
using *stb-sb-statfs*
by(*simp add*: *valid-def*)

lemma *security-sb-mount-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s\ .\ s = sa|\!\}$ *security-sb-mount s dev-name path type flgs data*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-sb-mount-def*
using *stb-sb-mount*
by *simp*

lemma *security-sb-umount-notchgstate* :
$\bigwedge sa$. $\{\!|\lambda s\ .\ s = sa|\!\}$ *security-sb-umount s vmnt flgs*
$\{\!|\lambda r\ s.\ s = sa|\!\}$
unfolding *security-sb-umount-def*
using *stb-sb-umount*

by *simp*

lemma *security-sb-pivotroot-notchgstate* :
$\bigwedge sa$. $\{\!\mid \lambda s$. $s = sa \mid\!\}$ *security-sb-pivotroot s old-path new-path*
$\{\!\mid \lambda r\ s.\ s = sa \mid\!\}$
unfolding *security-sb-pivotroot-def*
using *stb-sb-pivotroot*
by *simp*

lemma *security-sb-set-mnt-opts-notchgstate* :
$\bigwedge sa$. $\{\!\mid \lambda s$. $s = sa \mid\!\}$*security-sb-set-mnt-opts s sb opt kern-flags set-kern-flags*
$\{\!\mid \lambda r\ s.\ s = sa \mid\!\}$
unfolding *security-sb-set-mnt-opts-def*
using *stb-sb-set-mnt-opts*
by *simp*

end

27.9 init lsm hooks func

definition *security-binder-set-context-mgr'*:: $'s \Rightarrow Task \Rightarrow ('s, int)$ *nondet-monad*
where *security-binder-set-context-mgr' s mgr* $\equiv$ *do*
$r \leftarrow (return\ 0)$;
$return(r)$
od

lemma *binder-set-context-mgr'*:$\{\!\mid \lambda s.\ True \mid\!\}$ *security-binder-set-context-mgr' s t* $\{\!\mid \lambda r$
$s.\ r = 0\ \mid\!\}$
apply(*simp add* :*security-binder-set-context-mgr'-def*)
apply *wpsimp*
done

definition *security-binder-transaction'*:: $'s \Rightarrow Task \Rightarrow Task \Rightarrow ('s, int)$ *nondet-monad*
where *security-binder-transaction' s from to* $\equiv$ *do*
$r \leftarrow (return\ 0)$;
$return(r)$
od

definition *security-binder-transfer-binder'*:: $'s \Rightarrow Task \Rightarrow Task \Rightarrow ('s, int)$ *nondet-monad*
where *security-binder-transfer-binder' s from to* $\equiv$ *do*
$r \leftarrow (return\ 0)$;
$return(r)$
od

definition *security-binder-transfer-file'*:: $'s \Rightarrow Task \Rightarrow Task \Rightarrow Files \Rightarrow ('s, int)$
nondet-monad
where *security-binder-transfer-file' s from to file* $\equiv$ *do*
$r \leftarrow (return\ 0)$;
$return(r)$

od

definition *security-syslog′*:: *′s* ⇒ *int* ⇒(*′s*, *int*) *nondet-monad*
  where *security-syslog′ s type′* ≡ *do*
      r ← (*return 0*);
      return(*r*)
    od

definition *security-bprm-check′*:: *′s* ⇒ *linux-binprm* ⇒(*′s*, *int*) *nondet-monad*
  where *security-bprm-check′ s bprm* ≡ *do*
      r ← (*return 0*);
      return(*r*)
    od

definition *security-bprm-committing-creds′*:: *′s* ⇒ *linux-binprm* ⇒(*′s*, *unit*) *nondet-monad*
  where *security-bprm-committing-creds′ s bprm* ≡ *return*()

definition *security-bprm-committed-creds′*:: *′s* ⇒ *linux-binprm* ⇒(*′s*, *unit*) *nondet-monad*
  where *security-bprm-committed-creds′ s bprm* ≡ *do*
      r ← (*return bprm*);
      return()
    od

definition *security-sb-alloc′* :: *′s* ⇒ *super-block* ⇒(*′s*, *int*) *nondet-monad*
  where *security-sb-alloc′ s sb* ≡ *do*
      r ← (*return 0*);
      return(*r*)
    od

definition *security-sb-free′* :: *′s* ⇒ *super-block* ⇒(*′s*, *unit*) *nondet-monad*
  where *security-sb-free′ s sb* ≡ *return*()

definition *security-sb-copy-data′* :: *′s* ⇒ *string* ⇒ *string* ⇒(*′s*, *int*) *nondet-monad*
  where *security-sb-copy-data′ s orig copy* ≡ *do*
      r ← (*return 0*);
      return(*r*)
    od

definition *security-sb-remount′* :: *′s* ⇒ *super-block* ⇒ *Void* ⇒(*′s*, *int*) *nondet-monad*
  where *security-sb-remount′ s sb data* ≡ *do*
      r ← (*return 0*);
      return(*r*)
    od

definition *security-sb-kern-mount′* :: *′s* ⇒ *super-block* ⇒ *int* ⇒ *Void*⇒(*′s*, *int*) *nondet-monad*

where *security-sb-kern-mount′ s sb flag data* ≡ *do*
r ← (*return 0*);
return(*r*)
od

definition *security-sb-show-options′* :: *′s* ⇒*seq-file* ⇒*super-block* ⇒(*′s*, *int*) *nondet-monad*
where *security-sb-show-options′ s m sb* ≡ *do*
r ← (*return 0*);
return(*r*)
od

definition *security-sb-statfs′* :: *′s* ⇒ *dentry* ⇒(*′s*, *int*) *nondet-monad*
where *security-sb-statfs′ s dentry* ≡ *do*
r ← (*return 0*);
return(*r*)
od

definition *security-sb-mount′* :: *′s* ⇒ *string* ⇒ *path* ⇒*string* ⇒*int* ⇒ *Void* ⇒(*′s*, *int*) *nondet-monad*
where *security-sb-mount′ s devname path type flag data* ≡ *do*
r ← (*return 0*);
return(*r*)
od

definition *security-sb-umount′* :: *′s* ⇒ *vfsmount* ⇒ *int* ⇒(*′s*, *int*) *nondet-monad*
where *security-sb-umount′ s mnt′ flag* ≡ *do*
r ← (*return 0*);
return(*r*)
od

definition *security-sb-pivotroot′* :: *′s* ⇒ *path* ⇒ *path* ⇒(*′s*, *int*) *nondet-monad*
where *security-sb-pivotroot′ s old new* ≡ *do*
r ← (*return 0*);
return(*r*)
od

definition *security-sb-set-mnt-opts′* :: *′s*⇒*super-block* ⇒ *opts* ⇒ *int* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-sb-set-mnt-opts′ s sb opts′ kflags set-kflags* ≡ *do*
r ← (*return 0*);
return(*r*)
od

definition *security-sb-clone-mnt-opts′* :: *′s*⇒*super-block* ⇒ *super-block* ⇒ *int* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-sb-clone-mnt-opts′ s oldsb newsb kflags set-kflags* ≡ *do*
r ← (*return 0*);
return(*r*)
od

definition *security-sb-parse-opts-str'* :: *'s⇒string ⇒ opts ⇒ ('s, int) nondet-monad*
  where *security-sb-parse-opts-str' s options opt* ≡ *do*
    r ← (return 0);
    return(r)
  od

definition *security-task-alloc'* :: *'s⇒Task ⇒nat⇒ ('s, int) nondet-monad*
  where *security-task-alloc' s p f* ≡ *do*
    r ← (return 0);
    return(r)
    od

definition *security-task-free'* :: *'s⇒Task⇒ ('s, unit) nondet-monad*
  where *security-task-free' s p* ≡ *return()*

definition *security-task-fix-setuid'* :: *'s⇒Cred ⇒Cred ⇒int⇒ ('s, int) nondet-monad*
  where *security-task-fix-setuid' s new old f*≡ *do*
    r ← (return 0);
    return(r)
    od
definition *security-task-prlimit'* :: *'s⇒Cred ⇒Cred ⇒int⇒ ('s, int) nondet-monad*
  where *security-task-prlimit' s c tc f*≡ *do*
    r ← (return 0);
    return(r)
    od

definition *security-task-setrlimit'* :: *'s⇒ Task ⇒nat ⇒rlimit⇒ ('s, int) nondet-monad*
  where *security-task-setrlimit' s p r f*≡ *do*
    r ← (return 0);
    return(r)
    od

definition *security-task-prctl'* :: *'s⇒int ⇒nat ⇒nat ⇒nat ⇒nat⇒ ('s, int) nondet-monad*
  where *security-task-prctl' s op arg2 arg3 arg4 arg5*≡ *do*
    r ← (return 0);
    return(r)
    od

lemma *l-security-sb-alloc*:⦃λ*s. True*⦄ *security-sb-alloc' s sb* ⦃λ*r s. r = 0* ⦄
  apply(*simp add* :*security-sb-alloc'-def*)
  apply *wpsimp*
  done

lemma *l-security-sb-copy-data*:⦃λ*s. True*⦄ *security-sb-copy-data' s orig copy* ⦃λ*r*
s. r = 0 ⦄
  apply(*simp add* :*security-sb-copy-data'-def*)
  apply *wpsimp*
  done

lemma *l-security-sb-set-mnt-opts*:$\{\!|\lambda s.\ True|\!\}$ *security-sb-set-mnt-opts′ s sb opts′ kflags set-kflags* $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
  apply(*simp add* :*security-sb-set-mnt-opts′-def*)
  apply *wpsimp*
  done

definition *security-capget′* :: $'s \Rightarrow Task \Rightarrow kct \Rightarrow kct \Rightarrow kct \Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-capget′ s target effective inheritable permitted* ≡ *return 0*

definition *security-capset′* :: $'s \Rightarrow Cred \Rightarrow Cred \Rightarrow kct \Rightarrow kct \Rightarrow kct \Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-capset′ s new old effective inheritable permitted* ≡ *return 0*

definition *security-capable′* :: $'s \Rightarrow Cred \Rightarrow ns \Rightarrow cap \Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-capable′ s c ns cap* ≡ *return 0*

definition *security-capable-noaudit′* :: $'s \Rightarrow Cred \Rightarrow ns \Rightarrow cap \Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-capable-noaudit′ s c ns cap* ≡ *return 0*

definition *security-quotactl′*:: $'s \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow$ *super-block option* $\Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-quotactl′ s cmds t id′ sb* ≡ *return 0*

definition *security-quota-on′*:: $'s \Rightarrow dentry \Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-quota-on′ s dentry* ≡ *return 0*

definition *security-settime64′* :: $'s \Rightarrow ts \Rightarrow tz$ *option* $\Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-settime64′ s ts tz* ≡ *return 0*

definition *security-vm-enough-memory-mm′* :: $'s \Rightarrow mm \Rightarrow pages \Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-vm-enough-memory-mm′ s mm′ pages* ≡ *return 0*

definition *security-bprm-set-creds′* :: $'s \Rightarrow$ *linux-binprm* $\Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-bprm-set-creds′ s bprm* ≡ *return 0*

definition *security-dentry-init-security′* :: $'s \Rightarrow dentry \Rightarrow mode \Rightarrow string \Rightarrow string \Rightarrow int \Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-dentry-init-security′ s dentry m name ctx xtxlen* ≡ *return 0*

definition *security-dentry-create-files-as′* :: $'s \Rightarrow dentry \Rightarrow mode \Rightarrow string \Rightarrow Cred \Rightarrow Cred \Rightarrow ('s,\ int)$ *nondet-monad*
  where *security-dentry-create-files-as′ s dentry m name old new* ≡ *return 0*

definition *security-d-instantiate′*:: $'s \Rightarrow dentry \Rightarrow inode \Rightarrow ('s,\ unit)$ *nondet-monad*
  where *security-d-instantiate′ s dentry inode* ≡ *return* ()

definition *security-getprocattr'* :: *'s* ⇒ *Task* ⇒ *string* ⇒ *string* ⇒ (*'s*, *int*) *nondet-monad*
where *security-getprocattr' s p name value* ≡ *return 0*

definition *security-setprocattr'* :: *'s* ⇒ *string* ⇒ *string* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
where *security-setprocattr' s name value size'* ≡ *return 0*

definition *security-inode-alloc'* :: *'s* ⇒*inode* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-alloc' s inode* ≡ *return 0*

definition *security-inode-free'* :: *'s* ⇒ *inode* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-inode-free' s inode* ≡ *return*()

definition *security-inode-init-security'* :: *'s*⇒*inode* ⇒ *inode* ⇒ *string* ⇒ *string* ⇒ *string* ⇒*int* ⇒(*'s*, *int*) *nondet-monad*
where *security-inode-init-security' s inode dir qstr name value len'* = *return 0*

definition *security-old-inode-init-security'* :: *'s* ⇒ *inode* ⇒ *inode* ⇒ *qstr* ⇒ *string* ⇒ *string* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
where *security-old-inode-init-security' s inode dir qstr name value len'* ≡ *return 0*

definition *security-inode-create'* :: *'s* ⇒ *inode* ⇒ *dentry*⇒ *mode* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-create' s dir dentry m* ≡ *return 0*

definition *security-inode-link'* :: *'s* ⇒ *dentry* ⇒ *inode* ⇒ *dentry* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-link' s old-dentry dir new-dentry* ≡ *return 0*

definition *security-inode-unlink'* :: *'s* ⇒ *inode* ⇒ *dentry* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-unlink' s dir dentry* ≡ *return 0*

definition *security-inode-symlink'* :: *'s* ⇒ *inode* ⇒ *dentry* ⇒ *string* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-symlink' s dir dentry old-name* ≡ *return 0*

definition *security-inode-mkdir'* :: *'s* ⇒ *inode* ⇒ *dentry*⇒ *mode* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-mkdir' s dir dentry m* ≡ *return 0*

definition *security-inode-rmdir'* :: *'s* ⇒ *inode* ⇒ *dentry*⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-rmdir' s dir dentry* ≡ *return 0*

definition *security-inode-mknod'* :: *'s* ⇒ *inode* ⇒ *dentry* ⇒ *mode* ⇒ *dev-t*⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-mknod' s dir dentry m dev* ≡ *return 0*

definition *security-inode-rename'* :: *'s* ⇒ *inode* ⇒ *dentry* ⇒ *inode* ⇒*dentry* ⇒(*'s*, *int*) *nondet-monad*
where *security-inode-rename' s old-dir old-dentry new-dir new-dentry* ≡ *return 0*

definition *security-inode-readlink'* :: *'s* ⇒ *dentry* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-readlink' s dentry* ≡ *return 0*

definition *security-inode-follow-link'* :: *'s* ⇒ *dentry* ⇒ *inode* ⇒ *bool* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-follow-link' s dentry inode rcu'* ≡ *return 0*

definition *security-inode-permission'* :: *'s* ⇒ *inode* ⇒ *mask* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-permission' s inode m* ≡ *return 0*

definition *security-inode-setattr'* :: *'s* ⇒ *dentry* ⇒ *iattr* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-setattr' s dentry attr'* ≡ *return 0*

definition *security-inode-getattr'* :: *'s* ⇒ *path* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-getattr' s path* ≡ *return 0*

definition *security-inode-setxattr'* :: *'s* ⇒ *dentry* ⇒ *xattr* ⇒ *string* ⇒ *int* ⇒ *flags* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-setxattr' s dentry name value size' flgs* ≡ *return 0*

definition *evm-inode-post-setxattr'* :: *'s* ⇒ *dentry* ⇒*xattr* ⇒*string* ⇒*int* ⇒ (*'s*, *unit*) *nondet-monad*
where *evm-inode-post-setxattr' s d x value flg* ≡ *return* ()

definition *security-inode-post-setxattr'* :: *'s* ⇒ *dentry* ⇒ *xattr* ⇒ *string* ⇒ *int* ⇒ *flags* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-inode-post-setxattr' s dentry name value size' flgs* ≡ *return* ()

definition *security-inode-getxattr'* :: *'s* ⇒ *dentry* ⇒ *xattr* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-getxattr' s dentry name* ≡ *return 0*

definition *security-inode-listxattr'* :: *'s* ⇒ *dentry* ⇒ (*'s*, *int*) *nondet-monad*
where *security-inode-listxattr' s dentry* ≡ *return 0*

definition *security-inode-removexattr'* :: *'s* ⇒ *dentry* ⇒ *xattr* ⇒ (*'s*, *int*) *nondet-monad*

where *security-inode-removexattr' s dentry name* $\equiv$ *return 0*

definition *security-inode-need-killpriv'* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-need-killpriv' s dentry* $\equiv$ *return 0*

definition *security-inode-killpriv'* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-killpriv' s dentry* $\equiv$ *return 0*

definition *security-inode-getsecurity'* :: *'s* $\Rightarrow$ *inode* $\Rightarrow$ *xattr* $\Rightarrow$ *Void* $\Rightarrow$ *bool* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-getsecurity' s inode name buffer alloc* $\equiv$ *return* ($-$*EOPNOTSUPP*)

definition *security-inode-setsecurity'* :: *'s* $\Rightarrow$ *inode* $\Rightarrow$ *xattr* $\Rightarrow$ *Void* $\Rightarrow$ *nat*$\Rightarrow$ *int* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-setsecurity' s inode name value size' flgs* $\equiv$ *return* ($-$*EOPNOTSUPP*)

definition *security-inode-listsecurity'* :: *'s* $\Rightarrow$ *inode* $\Rightarrow$ *Void* $\Rightarrow$ *int* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-listsecurity' s inode buffer bsize* $\equiv$
return 0

definition *security-inode-getsecid'* :: *'s* $\Rightarrow$ *inode* $\Rightarrow$ *u32* $\Rightarrow$ (*'s*, *unit*) *nondet-monad*
where *security-inode-getsecid' s inode secid'* $\equiv$ *return*()

definition *security-inode-copy-up'*:: *'s* $\Rightarrow$ *dentry* $\Rightarrow$ *Cred option* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-copy-up' s src new* $\equiv$ *return 0*

definition *security-inode-copy-up-xattr'* :: *'s* $\Rightarrow$ *string* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-copy-up-xattr' s name* $\equiv$ *return 0*

definition *security-inode-invalidate-secctx'* :: *'s* $\Rightarrow$ *inode* $\Rightarrow$ (*'s*, *unit*) *nondet-monad*
where *security-inode-invalidate-secctx' s inode* $\equiv$*return* ()

definition *security-inode-notifysecctx'* :: *'s* $\Rightarrow$ *inode* $\Rightarrow$ *string* $\Rightarrow$ *u32*$\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-notifysecctx' s inode ctx ctxlen* $\equiv$ *return 0*

definition *security-inode-setsecctx'* :: *'s* $\Rightarrow$ *dentry* $\Rightarrow$ *string* $\Rightarrow$ *u32*$\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-setsecctx' s dentry ctx ctxlen* $\equiv$ *return 0*

definition *security-inode-getsecctx'* :: *'s* $\Rightarrow$ *inode* $\Rightarrow$ *string* $\Rightarrow$ *u32*$\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-inode-getsecctx' s dentry ctx ctxlen* $\equiv$ *return 0*

definition *security-file-permission'* :: *'s* $\Rightarrow$ *Files*$\Rightarrow$ *int* $\Rightarrow$ (*'s*, *int*) *nondet-monad*
where *security-file-permission' s file mask'* $\equiv$ *return 0*

definition *security-file-alloc′* :: *′s* ⇒ *Files* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-alloc′ s file* ≡ *return 0*

definition *security-file-free′* :: *′s* ⇒ *Files* ⇒ (*′s*, *unit*) *nondet-monad*
where *security-file-free′ s file* ≡ *return* ()

definition *security-file-ioctl′* :: *′s* ⇒ *Files*⇒ *IOC-DIR* ⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-ioctl′ s file cmd arg* ≡ *return 0*

definition *security-mmap-file′* :: *′s* ⇒ *Files option*⇒ *nat* ⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-mmap-file′ s file prot flgs* ≡ *return 0*

definition *security-mmap-addr′* :: *′s* ⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-mmap-addr′ s addr* ≡ *return 0*

definition *security-file-mprotect′* :: *′s* ⇒ *vm-area-struct*⇒ *nat* ⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-mprotect′ s vma reqprot prot* ≡ *return 0*

definition *security-file-lock′* :: *′s* ⇒ *Files*⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-lock′ s file cmd* ≡ *return 0*

definition *security-file-fcntl′* :: *′s* ⇒ *Files*⇒ *nat* ⇒ *nat* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-fcntl′ s file cmd arg* ≡ *return 0*

definition *security-file-set-fowner′* :: *′s* ⇒ *Files*⇒ (*′s*, *unit*) *nondet-monad*
where *security-file-set-fowner′ s file* ≡ *return* ()

definition *security-file-send-sigiotask′* :: *′s* ⇒ *Task*⇒ *fown-struct* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-file-send-sigiotask′ s tsk′ fown sig* ≡ *return 0*

definition *security-file-receive′* :: *′s* ⇒ *Files*⇒ (*′s*, *int*) *nondet-monad*
where *security-file-receive′ s file* ≡ *return 0*

definition *security-file-open′* :: *′s* ⇒ *Files*⇒ (*′s*, *int*) *nondet-monad*
where *security-file-open′ s file* ≡ *return 0*

definition *security-kernel-act-as′* :: *′s* ⇒ *Cred* ⇒ *u32* ⇒ (*′s*, *int*) *nondet-monad*
where *security-kernel-act-as′ s new secid′* ≡ *return 0*

definition *security-kernel-create-files-as′* :: *′s* ⇒ *Cred* ⇒ *inode* ⇒ (*′s*, *int*) *nondet-monad*
where *security-kernel-create-files-as′ s new inode* ≡ *return 0*

definition *security-kernel-module-request'* :: *'s* ⇒ *string* ⇒ (*'s*, *int*) *nondet-monad*
where *security-kernel-module-request' s name* ≡ *return 0*

definition *security-kernel-load-data'* :: *'s* ⇒ *kernel-load-data-id* ⇒ (*'s*, *int*) *nondet-monad*
where *security-kernel-load-data' s kid* ≡ *return 0*

definition *security-kernel-read-file'* :: *'s* ⇒ *Files* ⇒*kernel-read-file-id* ⇒ (*'s*, *int*) *nondet-monad*
where *security-kernel-read-file' s file kid* ≡ *return 0*

definition *security-kernel-post-read-file'* :: *'s* ⇒ *Files* ⇒ *string* ⇒ *nat* ⇒*kernel-read-file-id* ⇒ (*'s*, *int*) *nondet-monad*
where *security-kernel-post-read-file' s file buf size' kid* ≡ *return 0*

definition *security-ipc-permission'* :: *'s* ⇒ *kern-ipc-perm* ⇒ *nat* ⇒ (*'s*, *int*) *nondet-monad*
where *security-ipc-permission' s ipcp flg* ≡ *return 0*

definition *security-ipc-getsecid'* :: *'s* ⇒ *kern-ipc-perm* ⇒ *u32* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-ipc-getsecid' s ipcp secid'* ≡ *return* ()

definition *security-msg-msg-alloc'* :: *'s* ⇒ *msg-msg* ⇒ (*'s*, *int*) *nondet-monad*
where *security-msg-msg-alloc' s msg* ≡ *return 0*

definition *security-msg-msg-free'* :: *'s* ⇒ *msg-msg* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-msg-msg-free' s msg* ≡ *return* ()

definition *security-msg-queue-alloc'* :: *'s* ⇒ *kern-ipc-perm* ⇒ (*'s*, *int*) *nondet-monad*
where *security-msg-queue-alloc' s msq* ≡ *return 0*

definition *security-msg-queue-free'* :: *'s* ⇒ *kern-ipc-perm* ⇒ (*'s*, *int*) *nondet-monad*
where *security-msg-queue-free' s msq* ≡ *return 0*

definition *security-msg-queue-associate'* :: *'s* ⇒ *kern-ipc-perm* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
where *security-msg-queue-associate' s msq msqflg* ≡ *return 0*

definition *security-msg-queue-msgctl'* :: *'s* ⇒ *kern-ipc-perm* ⇒ *IPC-CMD*⇒ (*'s*, *int*) *nondet-monad*
where *security-msg-queue-msgctl' s msq cmd* ≡ *return 0*

definition *security-msg-queue-msgsnd'* :: *'s* ⇒ *kern-ipc-perm* ⇒ *msg-msg* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
where *security-msg-queue-msgsnd' s msq msg msqflg* ≡ *return 0*

definition *security-msg-queue-msgrcv'* :: *'s* ⇒ *kern-ipc-perm* ⇒ *msg-msg*⇒ *Task* ⇒ *int* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
where *security-msg-queue-msgrcv' s msq msg target type m* ≡ *return 0*

definition *security-shm-alloc'* :: *'s* ⇒ *kern-ipc-perm* ⇒ (*'s*, *int*) *nondet-monad*
where *security-shm-alloc' s shp* ≡ *return 0*

definition *security-shm-free'* :: *'s* ⇒ *kern-ipc-perm* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-shm-free' s shp* ≡ *return* ()

definition *security-shm-associate'* :: *'s* ⇒ *kern-ipc-perm* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
where *security-shm-associate' s shp shmflg* ≡ *return 0*

definition *security-shm-shmctl'* :: *'s* ⇒ *kern-ipc-perm* ⇒ *IPC-CMD*⇒ (*'s*, *int*) *nondet-monad*
where *security-shm-shmctl' s shp cmd* ≡ *return 0*

definition *security-shm-shmat'* :: *'s* ⇒ *kern-ipc-perm* ⇒*string* ⇒*int* ⇒ (*'s*, *int*) *nondet-monad*
where *security-shm-shmat' s shp shmaddr shmflg* ≡ *return 0*

definition *security-sem-alloc'* :: *'s* ⇒ *kern-ipc-perm* ⇒ (*'s*, *int*) *nondet-monad*
where *security-sem-alloc' s sma* ≡ *return 0*

definition *security-sem-free'* :: *'s* ⇒ *kern-ipc-perm* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-sem-free' s sma* ≡ *return* ()

definition *security-sem-associate'*:: *'s* ⇒ *kern-ipc-perm* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
where *security-sem-associate' s sma semflg* ≡ *return 0*

definition *security-sem-semctl'* :: *'s* ⇒ *kern-ipc-perm* ⇒ *IPC-CMD*⇒ (*'s*, *int*) *nondet-monad*
where *security-sem-semctl' s sma cmd* ≡ *return 0*

definition *security-sem-semop'* :: *'s* ⇒ *kern-ipc-perm* ⇒*sembuf*⇒*nat* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
where *security-sem-semop' s sma sops nsops alter* ≡ *return 0*

definition *security-netlink-send'* :: *'s* ⇒ *sock* ⇒ *sk-buff*⇒ (*'s*, *int*) *nondet-monad*
where *security-netlink-send' s sk' skb* ≡ *return 0*

definition *security-ismaclabel'* :: *'s* ⇒ *xattr*⇒ (*'s*, *int*) *nondet-monad*
where *security-ismaclabel' s name* ≡ *return 0*

definition *security-secid-to-secctx'* :: *'s* ⇒ *u32* ⇒*string* ⇒ *u32* ⇒ (*'s*, *int*) *nondet-monad*
where *security-secid-to-secctx' s secid' secdata seclen* ≡ *return 0*

definition *security-secctx-to-secid′* :: *′s* ⇒ *string* ⇒ *u32*⇒ *u32*⇒ (*′s*, *int*) *nondet-monad*
where *security-secctx-to-secid′ s secdata seclen secid′* ≡*return 0*

definition *security-release-secctx′* :: *′s* ⇒ *string* ⇒ *u32*⇒ (*′s*, *unit*) *nondet-monad*
where *security-release-secctx′ s secdata seclen* ≡ *return* ()

definition *security-unix-stream-connect′* :: *′s* ⇒ *sock* ⇒ *sock* ⇒ *sock* ⇒ (*′s*, *int*) *nondet-monad*
where*security-unix-stream-connect′ s sock other newsk* ≡ *return 0*

definition *security-unix-may-send′* :: *′s* ⇒ *socket* ⇒ *socket* ⇒ (*′s*, *int*) *nondet-monad*
where *security-unix-may-send′ s sock other* ≡ *return 0*

definition *security-socket-create′* :: *′s* ⇒ *Sk-Family* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-create′ s family type pro kern* ≡ *return 0*

definition *security-socket-post-create′* :: *′s* ⇒ *socket* ⇒ *Sk-Family* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-post-create′ s sock family type pro kern* ≡ *return 0*

definition *security-socket-socketpair′* :: *′s* ⇒ *socket* ⇒ *socket* ⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-socketpair′ s socka sockb* ≡ *return 0*

definition *security-socket-bind′* :: *′s* ⇒ *socket* ⇒ *sockaddr* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-bind′ s sock address addrlen* ≡ *return 0*

definition *security-socket-connect′* :: *′s* ⇒ *socket* ⇒ *sockaddr* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-connect′ s sock address addrlen* ≡ *return 0*

definition *security-socket-listen′* :: *′s* ⇒ *socket* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-listen′ s sock backlog* ≡ *return 0*

definition *security-socket-accept′* :: *′s* ⇒ *socket* ⇒ *socket* ⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-accept′ s sock newsock* ≡ *return 0*

definition *security-socket-sendmsg′* :: *′s* ⇒ *socket* ⇒ *msghdr* ⇒ *int* ⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-sendmsg′ s sock msg size′* ≡ *return 0*

definition *security-socket-recvmsg′* :: *′s* ⇒ *socket* ⇒ *msghdr* ⇒ *int* ⇒ *int*⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-recvmsg′ s sock msg size′ flgs* ≡ *return 0*

definition *security-socket-getsockname′* :: *′s* ⇒ *socket* ⇒ (*′s*, *int*) *nondet-monad*
where *security-socket-getsockname′ s sock* ≡ *return 0*

definition *security-socket-getpeername'* :: *'s* ⇒ *socket* ⇒ (*'s*, *int*) *nondet-monad*
where *security-socket-getpeername' s sock* ≡ *return 0*

definition *security-socket-getsockopt'* :: *'s* ⇒ *socket* ⇒*int* ⇒ *int*⇒ (*'s*, *int*) *nondet-monad*
where *security-socket-getsockopt' s sock level' optname* ≡ *return 0*

definition *security-socket-setsockopt'* :: *'s* ⇒ *socket* ⇒*int* ⇒ *int* ⇒ (*'s*, *int*) *nondet-monad*
where *security-socket-setsockopt' s sock level' optname* ≡ *return 0*

definition *security-socket-shutdown'*:: *'s* ⇒ *socket* ⇒*int* ⇒ (*'s*, *int*) *nondet-monad*
where *security-socket-shutdown' s sock how* ≡ *return 0*

definition *security-sock-rcv-skb'* :: *'s* ⇒ *sock* ⇒*sk-buff* ⇒ (*'s*, *int*) *nondet-monad*
where *security-sock-rcv-skb' s sock skb* ≡ *return 0*

definition *security-socket-getpeersec-stream'* :: *'s* ⇒ *socket* ⇒*string* ⇒*int* ⇒*nat*
⇒ (*'s*, *int*) *nondet-monad*
where *security-socket-getpeersec-stream' s sock optval optlen len'* ≡ *return 0*

definition *security-socket-getpeersec-dgram'* :: *'s* ⇒ *socket* ⇒*sk-buff option* ⇒ *u32*
⇒ (*'s*, *int*) *nondet-monad*
where *security-socket-getpeersec-dgram' s sock skb secid'* ≡ *return 0*

definition *security-sk-alloc'* :: *'s* ⇒ *sock*⇒*int* ⇒ *gfp-t* ⇒ (*'s*, *int*) *nondet-monad*
where *security-sk-alloc' s sk' family priority* ≡ *return 0*

definition *security-sk-free'* :: *'s* ⇒ *sock*⇒(*'s*, *unit*) *nondet-monad*
where *security-sk-free' s sock* ≡ *return* ()

definition *security-sk-clone'* :: *'s* ⇒ *sock*⇒ *sock* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-sk-clone' s sk' newsk* ≡ *return*()

definition *security-sk-classify-flow'* :: *'s* ⇒ *sock*⇒ *flowi* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-sk-classify-flow' s sock' fl* ≡ *return* ()

definition *security-req-classify-flow'* :: *'s* ⇒ *request-sock*⇒ *flowi* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-req-classify-flow' s req fl* ≡ *return* ()

definition *security-sock-graft'* :: *'s* ⇒ *sock*⇒ *socket* ⇒ (*'s*, *unit*) *nondet-monad*
where *security-sock-graft' s sk' parent'* ≡ *return* ()

definition *security-inet-conn-request'* :: *'s* ⇒ *sock*⇒ *sk-buff* ⇒*request-sock*
⇒ (*'s*, *int*) *nondet-monad*
where *security-inet-conn-request' s sk' skb req* ≡ *return 0*

definition *security-inet-csk-clone'* :: *'s* ⇒ *sock* ⇒*request-sock* ⇒ (*'s*, *unit*) *nondet-monad*
  where *security-inet-csk-clone' s newsk req* ≡ *return* ()

definition *security-inet-conn-established'* :: *'s* ⇒ *sock*⇒ *sk-buff* ⇒ (*'s*, *unit*) *nondet-monad*
  where *security-inet-conn-established' s sk' skb* ≡ *return*()

definition *security-secmark-relabel-packet'* :: *'s* ⇒ *u32* ⇒ (*'s*, *int*) *nondet-monad*
  where *security-secmark-relabel-packet' s secid'* ≡ *return 0*

definition *security-secmark-refcount-inc* :: *'s* ⇒ (*'s*, *unit*) *nondet-monad*
  where *security-secmark-refcount-inc s* ≡ *return* ()

definition *security-secmark-refcount-dec'* :: *'s* ⇒ (*'s*, *unit*) *nondet-monad*
  where*security-secmark-refcount-dec' s* ≡ *return* ()

definition *security-tun-dev-alloc-security'* :: *'s* ⇒ *'b*⇒(*'s*, *int*) *nondet-monad*
  where *security-tun-dev-alloc-security' s security* ≡ *return 0*

definition *security-tun-dev-free-security'* :: *'s* ⇒ *'b* ⇒(*'s*, *unit*) *nondet-monad*
  where *security-tun-dev-free-security' s security* ≡ *return* ()

definition *security-tun-dev-create* :: *'s* ⇒ (*'s*, *int*) *nondet-monad*
  where*security-tun-dev-create s* ≡ *return 0*

definition *security-tun-dev-attach-queue'* :: *'s* ⇒ *'b* ⇒(*'s*, *int*) *nondet-monad*
  where *security-tun-dev-attach-queue' s security* ≡ *return 0*

definition *security-tun-dev-attach'* :: *'s* ⇒ *sock*⇒ *'b* ⇒(*'s*, *int*) *nondet-monad*
  where *security-tun-dev-attach' s sk' security* ≡ *return 0*

definition *security-tun-dev-open'* :: *'s* ⇒ *'b* ⇒(*'s*, *int*) *nondet-monad*
  where*security-tun-dev-open' s security* ≡ *return 0*

definition *security-sctp-assoc-request'* :: *'s* ⇒ *sctp-endpoint* ⇒ *sk-buff* ⇒(*'s*, *int*) *nondet-monad*
  where *security-sctp-assoc-request' s ep skb* ≡ *return 0*

definition *security-sctp-bind-connect'* :: *'s* ⇒ *sock* ⇒ *int* ⇒ *sockaddr* ⇒*int* ⇒(*'s*, *int*) *nondet-monad*
  where *security-sctp-bind-connect' s sk' optname address addrlen* ≡ *return 0*

definition *security-sctp-sk-clone'* :: *'s* ⇒ *sctp-endpoint* ⇒ *sock* ⇒ *sock* ⇒(*'s*, *unit*) *nondet-monad*
  where *security-sctp-sk-clone' s ep sk' newsk* ≡ *return* ()

definition *security-ib-pkey-access'* :: *'s* ⇒ *'i* ⇒ *nat* ⇒ *nat* ⇒(*'s*, *int*) *nondet-monad*

where *security-ib-pkey-access′ s sec subnet-prefix pkey* ≡ *return 0*

definition *security-ib-endport-manage-subnet′*:: *′s* ⇒ *′i* ⇒ *string* ⇒ *nat* ⇒(*′s*, *int*) *nondet-monad*
where *security-ib-endport-manage-subnet′ s sec dev-name port-num* ≡ *return 0*

definition *security-ib-alloc-security′* :: *′s* ⇒ *′i list* ⇒ (*′s*, *int*) *nondet-monad*
where *security-ib-alloc-security′ s sec* ≡ *return 0*

definition *security-ib-free-security′* :: *′s* ⇒ *′i list* ⇒ (*′s*, *unit*) *nondet-monad*
where *security-ib-free-security′ s sec* ≡ *return* ()

definition *security-xfrm-policy-alloc′* :: *′s* ⇒ *xfrm-sec-ctx* ⇒ *xfrm-user-sec-ctx* ⇒ *gfp-t*
⇒ (*′s*, *int*) *nondet-monad*
where *security-xfrm-policy-alloc′ s ctxp sec-ctx gfp′* ≡ *return 0*

definition *security-xfrm-policy-clone′* :: *′s* ⇒ *xfrm-sec-ctx* ⇒ *xfrm-user-sec-ctx*
⇒ (*′s*, *int*) *nondet-monad*
where *security-xfrm-policy-clone′ s old-ctx new-ctxp* ≡ *return 0*

definition *security-xfrm-policy-free′* :: *′s* ⇒ *xfrm-sec-ctx* ⇒(*′s*, *unit*) *nondet-monad*
where *security-xfrm-policy-free′ s ctx* ≡ *return* ()

definition *security-xfrm-policy-delete′* :: *′s* ⇒ *xfrm-sec-ctx* ⇒(*′s*, *int*) *nondet-monad*
where *security-xfrm-policy-delete′ s ctx* ≡ *return 0*

definition *security-xfrm-state-alloc′* :: *′s* ⇒ *xfrm-state* ⇒ *xfrm-sec-ctx* ⇒(*′s*, *int*) *nondet-monad*
where *security-xfrm-state-alloc′ s x sec-ctx* ≡ *return 0*

definition *security-xfrm-state-alloc-acquire′* :: *′s* ⇒ *xfrm-state* ⇒ *xfrm-sec-ctx* ⇒ *u32*
⇒ (*′s*, *int*) *nondet-monad*
where *security-xfrm-state-alloc-acquire′ s x plosec secid′* ≡ *return 0*

definition *security-xfrm-state-delete′* :: *′s* ⇒ *xfrm-state* ⇒(*′s*, *int*) *nondet-monad*
where *security-xfrm-state-delete′ s x* ≡ *return 0*

definition *security-xfrm-state-free′* :: *′s* ⇒ *xfrm-state* ⇒(*′s*, *unit*) *nondet-monad*
where *security-xfrm-state-free′ s x* ≡ *return* ()

definition *security-xfrm-policy-lookup′* :: *′s* ⇒ *xfrm-sec-ctx* ⇒ *u32* ⇒ *u8* ⇒(*′s*, *int*) *nondet-monad*
where *security-xfrm-policy-lookup′ s ctx fl-secid dir* ≡ *return 0*

definition *security-xfrm-state-pol-flow-match′* :: *′s* ⇒ *xfrm-state* ⇒ *xfrm-policy* ⇒ *flowi*

$\Rightarrow$ ('s, int) nondet-monad
where *security-xfrm-state-pol-flow-match' s x xp fl* $\equiv$ *return 0*

definition *security-xfrm-decode-session'* :: 's $\Rightarrow$ *sk-buff* $\Rightarrow$ *u32* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-xfrm-decode-session' s skb secid'* $\equiv$ *return 0*

definition *security-skb-classify-flow'* :: 's $\Rightarrow$ *sk-buff* $\Rightarrow$ *flowi* $\Rightarrow$('s, *unit*) *nondet-monad*
where *security-skb-classify-flow' s skb fl* $\equiv$ *return* ()

definition *security-path-unlink'* :: 's $\Rightarrow$ *path* $\Rightarrow$ *dentry* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-unlink' s dir dentry* $\equiv$ *return 0*

definition *security-path-mkdir'* :: 's $\Rightarrow$ *path* $\Rightarrow$ *dentry* $\Rightarrow$*nat* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-mkdir' s dir dentry m* $\equiv$ *return 0*

definition *security-path-rmdir'* :: 's $\Rightarrow$ *path* $\Rightarrow$ *dentry* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-rmdir' s dir dentry* $\equiv$ *return 0*

definition *security-path-mknod'* :: 's $\Rightarrow$ *path* $\Rightarrow$ *dentry* $\Rightarrow$*nat* $\Rightarrow$ *nat*$\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-mknod' s dir dentry m dev* $\equiv$ *return 0*

definition *security-path-truncate'* :: 's $\Rightarrow$ *path* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-truncate' s dir* $\equiv$ *return 0*

definition *security-path-symlink'* :: 's $\Rightarrow$ *path* $\Rightarrow$ *dentry* $\Rightarrow$ *string* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-symlink' s dir dentry old-name* $\equiv$ *return 0*

definition *security-path-link'* :: 's $\Rightarrow$ *dentry* $\Rightarrow$*path* $\Rightarrow$ *dentry* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-link' s old-dentry new-dir new-dentry* $\equiv$ *return 0*

definition *security-path-rename'* :: 's $\Rightarrow$ *path* $\Rightarrow$ *dentry* $\Rightarrow$ *path* $\Rightarrow$ *dentry* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-rename' s old-dir old-dentry new-dir new-dentry* $\equiv$
return 0

definition *security-path-chmod'* :: 's $\Rightarrow$ *path* $\Rightarrow$ *nat*$\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-chmod' s path m* $\equiv$ *return 0*

definition *security-path-chown'* :: 's $\Rightarrow$ *path* $\Rightarrow$ *kuid-t* $\Rightarrow$ *kgid-t* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-chown' s path uid' gid'* $\equiv$ *return 0*

definition *security-path-chroot'* :: 's $\Rightarrow$ *path* $\Rightarrow$('s, *int*) *nondet-monad*
where *security-path-chroot' s path* $\equiv$ *return 0*

definition *security-bpf′* :: *′s* ⇒ *int* ⇒ *bpf-attr* ⇒ *nat* ⇒(*′s*, *int*) *nondet-monad*
where *security-bpf′ s cmd attr′ size′* ≡ *return 0*

definition *security-bpf-map′* :: *′s* ⇒ *bpf-map* ⇒ *mode* ⇒(*′s*, *int*) *nondet-monad*
where *security-bpf-map′ s bmap fmode* ≡ *return 0*

definition *security-bpf-prog′* :: *′s* ⇒ *bpf-prog* ⇒(*′s*, *int*) *nondet-monad*
where *security-bpf-prog′ s prog* ≡ *return 0*

definition *security-bpf-map-alloc′* :: *′s* ⇒ *bpf-map* ⇒(*′s*, *int*) *nondet-monad*
where *security-bpf-map-alloc′ s bmap* ≡ *return 0*

definition *security-bpf-map-free′* :: *′s* ⇒ *bpf-map* ⇒(*′s*, *unit*) *nondet-monad*
where *security-bpf-map-free′ s bmap* ≡ *return* ()

definition *security-bpf-prog-alloc′* :: *′s* ⇒ *bpf-prog-aux* ⇒(*′s*, *int*) *nondet-monad*
where *security-bpf-prog-alloc′ s aux′* ≡ *return 0*

definition *security-bpf-prog-free′* :: *′s* ⇒ *bpf-prog-aux* ⇒(*′s*, *unit*) *nondet-monad*
where *security-bpf-prog-free′ s aux′* ≡ *return* ()

definition *security-key-alloc′* :: *′s* ⇒ *key* ⇒ *Cred* ⇒ *nat* ⇒(*′s*, *int*) *nondet-monad*
where *security-key-alloc′ s key′ c flgs* ≡ *return 0*

definition *security-key-free′* :: *′s* ⇒ *key* ⇒(*′s*, *unit*) *nondet-monad*
where *security-key-free′ s key′* ≡ *return* ()

definition *security-key-permission′* :: *′s* ⇒ *key-ref-t* ⇒ *Cred* ⇒ *nat* ⇒(*′s*, *int*) *nondet-monad*
where *security-key-permission′ s key-ref c perm* ≡ *return 0*

definition *security-key-getsecurity′* :: *′s* ⇒ *key* ⇒ *string* ⇒(*′s*, *int*) *nondet-monad*
where *security-key-getsecurity′ s key′ buffer* ≡ *return 0*

definition *security-audit-rule-init′* :: *′s* ⇒ *nat* ⇒ *enum-audit* ⇒ *string* ⇒ *string*
⇒ (*′s*, *int*) *nondet-monad*
where *security-audit-rule-init′ s field op rulestr lsmrule* ≡ *return 0*

definition *security-audit-rule-known′* :: *′s* ⇒ *audit-krule* ⇒ (*′s*, *int*) *nondet-monad*
where *security-audit-rule-known′ s krule* ≡ *return 0*

definition *security-audit-rule-match′* :: *′s* ⇒ *nat* ⇒ *nat* ⇒ *enum-audit* ⇒ *string* ⇒ *audit-context*
⇒ (*′s*, *int*) *nondet-monad*
where *security-audit-rule-match′ s secid′ field op lsmrule actx* ≡ *return 0*

definition *security-audit-rule-free'* :: $'s \Rightarrow string \Rightarrow ('s, unit)$ *nondet-monad*
  where *security-audit-rule-free' s lsmrule* $\equiv$ *return* ()

end
theory *Dynamic-model*
imports *Main*
begin

27.10 Security State Machine

locale *SM* =
  fixes *s0* :: $'s$
  fixes *step* :: $'s \Rightarrow 'e \Rightarrow 's$
  fixes *domain* :: $'e \Rightarrow ('d\ option)$
  fixes *vpeq* :: $'s \Rightarrow 'd \Rightarrow 's \Rightarrow bool$ ((- ∼ - ∼ -))
  fixes *interferes* :: $'d \Rightarrow 's \Rightarrow 'd \Rightarrow bool$ ((- @ - -))
  assumes
    vpeq-transitive-lemma : $\forall\ s\ t\ r\ d.\ (s \sim d \sim t) \wedge (t \sim d \sim r) \longrightarrow (s \sim d \sim r)$
and
    vpeq-symmetric-lemma : $\forall\ s\ t\ d.\ (s \sim d \sim t) \longrightarrow (t \sim d \sim s)$ **and**
    vpeq-reflexive-lemma : $\forall\ s\ d.\ (s \sim d \sim s)$ **and**
    interf-reflexive: $\forall\, d\ s.\ (d\ @\ s\ \ d)$
begin

  definition *non-interferes* :: $'d \Rightarrow 's \Rightarrow 'd \Rightarrow bool$ ((- @ - \ -))
    where $(u\ @\ s \setminus v) \equiv (u\ @\ s\ \ v)$

  definition *ivpeq* :: $'s \Rightarrow 'd\ set \Rightarrow 's \Rightarrow bool$ ((- ≈ - ≈ -))
    where *ivpeq s D t* $\equiv \forall\ d \in D.\ (s \sim d \sim t)$

  primrec *run* :: $'s \Rightarrow 'e\ list \Rightarrow 's$
    where *run-Nil*: *run s* [] = *s* |
        run-Cons: *run s* (*a*#*as*) = *run* (*step s a*) *as*

  definition *reachable* :: $'s \Rightarrow 's \Rightarrow bool$ ((- ↪ -) [*70*,*71*] *60*) **where**
    reachable s1 s2 $\equiv$ ($\exists$ *as*. *run s1 as* = *s2*)

  definition *reachable0* :: $'s \Rightarrow bool$ **where**
    reachable0 s $\equiv$ *reachable s0 s*

  declare *non-interferes-def*[*cong*] **and** *ivpeq-def*[*cong*] **and** *reachable-def*[*cong*]
      and *reachable0-def*[*cong*] **and** *run.simps*(*1*)[*cong*] **and** *run.simps*(*2*)[*cong*]

  lemma *reachable-s0* : *reachable0 s0*
    by (*metis SM.reachable-def SM-axioms reachable0-def run.simps*(*1*))

  lemma *reachable-self* : *reachable s s*

using *reachable-def run.simps(1)* **by** *fastforce*

lemma *reachable-step* : $s' = step\ s\ a \Longrightarrow reachable\ s\ s'$
proof−
assume *a0*: $s' = step\ s\ a$
then have $s' = run\ s\ [a]$ **by** *auto*
then show *?thesis* **using** *reachable-def* **by** *blast*
qed

lemma *run-trans* : $\forall\, C\ T\ V\ as\ bs.\ T = run\ C\ as \wedge V = run\ T\ bs \longrightarrow V = run\ C\ (as@bs)$
proof −
{
fix $T\ V\ as\ bs$
have $\forall\, C.\ T = run\ C\ as \wedge V = run\ T\ bs \longrightarrow V = run\ C\ (as@bs)$
proof(*induct as*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons c cs*)
assume *a0*: $\forall\, C.\ T = run\ C\ cs \wedge V = run\ T\ bs \longrightarrow V = run\ C\ (cs\ @\ bs)$
show *?case*
proof−
{
fix C
have $T = run\ C\ (c\ \#\ cs) \wedge V = run\ T\ bs \longrightarrow V = run\ C\ ((c\ \#\ cs)\ @\ bs)$
proof
assume *b0*: $T = run\ C\ (c\ \#\ cs) \wedge V = run\ T\ bs$
from *b0* **obtain** C' **where** *b2*: $C' = step\ C\ c \wedge T = run\ C'\ cs$ **by** *auto*
with *a0 b0* **have** $V = run\ C'\ (cs@bs)$ **by** *blast*
with *b2* **show** $V = run\ C\ ((c\ \#\ cs)\ @\ bs)$
using *append-Cons run-Cons* **by** *auto*
qed
}
then show *?thesis* **by** *blast*
qed
qed
}
then show *?thesis* **by** *auto*
qed

lemma *reachable-trans* : $[\![reachable\ C\ T;\ reachable\ T\ V]\!] \Longrightarrow reachable\ C\ V$
proof−
assume *a0*: *reachable C T*
assume *a1*: *reachable T V*
from *a0* **have** $C = T \vee (\exists\, as.\ T = run\ C\ as)$ **by** *auto*
then show *?thesis*

```
        proof
          assume b0: C = T
          show ?thesis
            proof -
              from a1 have T = V ∨ (∃ as. V = run T as) by auto
              then show ?thesis
                proof
                  assume c0: T=V
                  with a0 show ?thesis by auto
                next
                  assume c0: (∃ as. V = run T as)
                  then show ?thesis using a1 b0 by auto
                qed
            qed
        next
          assume b0: ∃ as. T = run C as
          show ?thesis
            proof -
              from a1 have T = V ∨ (∃ as. V = run T as) by auto
              then show ?thesis
                proof
                  assume c0: T=V
                  then show ?thesis using a0 by auto
                next
                  assume c0: (∃ as. V = run T as)
                  from b0 obtain as where d0: T = run C as by auto
                  from c0 obtain bs where d1: V = run T bs by auto
                  then show ?thesis using d0 run-trans by fastforce
                qed
            qed
        qed
    qed

  lemma reachableStep : ⟦reachable0 C; C' = step C a⟧ ⟹ reachable0 C'
    apply (simp add: reachable0-def)
    using reachable-step reachable-trans by blast

  lemma reachable0-reach : ⟦reachable0 C; reachable C C'⟧ ⟹ reachable0 C'
    using reachable-trans by fastforce

  declare reachable-def[cong del] and reachable0-def[cong del]
end
```

27.11 Information flow security properties

```
locale SM-enabled = SM s0 step domain vpeq interferes
  for s0 :: 's and
      step :: 's ⇒ 'e ⇒ 's and
      domain :: 'e ⇒ ('d option) and
```

vpeq :: $'s \Rightarrow 'd \Rightarrow 's \Rightarrow$ *bool* ((- ∼ - ∼ -)) **and**
interferes :: $'d \Rightarrow 's \Rightarrow 'd \Rightarrow$ *bool* ((- @ - -))
+

assumes *enabled0*: ∀ *s a*. *reachable0 s* ⟶ (∃ *s'*. *s'* = *step s a*)
and *policy-respect*: ∀ *v u s t*. (*s* ∼ *u* ∼ *t*)
⟶ (*interferes v s u* = *interferes v t u*)

begin

lemma *enabled* : *reachable0 s* ⟹ (∃ *s'*. *s'* = *step s a*)
using *enabled0* **by** *simp*

primrec *sources* :: *'e list* ⇒ *'d* ⇒ *'s* ⇒ *'d set* **where**
sources-Nil:*sources* [] *d s* = {*d*} |
sources-Cons:*sources* (*a* # *as*) *d s* = (⋃{*sources as d* (*step s a*)}) ∪
{*w* . *w* = *the* (*domain a*) ∧ (∃ *v* . *interferes w s v* ∧
v∈*sources as d* (*step s a*))}
declare *sources-Nil* [*simp del*]
declare *sources-Cons* [*simp del*]

primrec *ipurge* :: *'e list* ⇒ *'d* ⇒ *'s* ⇒ *'e list* **where**
ipurge-Nil: *ipurge* [] *u s* = [] |
ipurge-Cons: *ipurge* (*a*#*as*) *u s* = (*if* (*the* (*domain a*) ∈ (*sources* (*a*#*as*) *u*
s))
then
a # *ipurge as u* (*step s a*)
else
ipurge as u (*step s a*)
)

definition *observ-equivalence* :: *'s* ⇒ *'e list* ⇒ *'s* ⇒
'e list ⇒ *'d* ⇒ *bool* ((- - ≅ - - @ -))
where *observ-equivalence s as t bs d* ≡
((*run s as*) ∼ *d* ∼ (*run t bs*))
declare *observ-equivalence-def*[*cong*]

lemma *observ-equiv-sym*:
(*s as* ≅ *t bs* @ *d*) ⟹ (*t bs* ≅ *s as* @ *d*)
using *observ-equivalence-def vpeq-symmetric-lemma* **by** *blast*

lemma *observ-equiv-trans*:
⟦*reachable0 t*; (*s as* ≅ *t bs* @ *d*); (*t bs* ≅ *x cs* @ *d*)⟧ ⟹ (*s as* ≅ *x cs*
@ *d*)
using *observ-equivalence-def vpeq-transitive-lemma* **by** *blast*

definition *noninterference-r* :: *bool*
where *noninterference-r* $\equiv \forall$ *d as s. reachable0 s* $\longrightarrow$ (*s* *as* $\cong$ *s* (*ipurge as d s*) @ *d*)

definition *noninterference* :: *bool*
where *noninterference* $\equiv \forall$ *d as.* (*s0* *as* $\cong$ *s0* (*ipurge as d s0*) @ *d*)

definition *weak-noninterference* :: *bool*
where *weak-noninterference* $\equiv \forall$ *d as bs. ipurge as d s0* = *ipurge bs d s0*
$\longrightarrow$ (*s0* *as* $\cong$ *s0* *bs* @ *d*)

definition *weak-noninterference-r* :: *bool*
where *weak-noninterference-r* $\equiv \forall$ *d as bs s. reachable0 s* $\wedge$ *ipurge as d s* = *ipurge bs d s*
$\longrightarrow$ (*s* *as* $\cong$ *s* *bs* @ *d*)

definition *noninfluence*::*bool*
where *noninfluence* $\equiv \forall$ *d as s t. reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\longrightarrow$ (*s* *as* $\cong$ *t* (*ipurge as d t*) @ *d*)

definition *weak-noninfluence* ::*bool*
where *weak-noninfluence* $\equiv \forall$ *d as bs s t . reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\wedge$ *ipurge as d t* = *ipurge bs d t*
$\longrightarrow$ (*s* *as* $\cong$ *t* *bs* @ *d*)

definition *weak-noninfluence2* ::*bool*
where *weak-noninfluence2* $\equiv \forall$ *d as bs s t . reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\wedge$ *ipurge as d s* = *ipurge bs d t*
$\longrightarrow$ (*s* *as* $\cong$ *t* *bs* @ *d*)

definition *nonleakage* :: *bool*
where *nonleakage* $\equiv \forall$ *d as s t. reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*) $\longrightarrow$ (*s* *as* $\cong$ *t* *as* @ *d*)

declare *noninterference-r-def*[*cong*] **and** *noninterference-def*[*cong*] **and** *weak-noninterference-def*[*cong*] **and**
weak-noninterference-r-def[*cong*] **and** *noninfluence-def*[*cong*] **and**
weak-noninfluence-def[*cong*] **and** *weak-noninfluence2-def*[*cong*] **and** *nonleakage-def*[*cong*]

27.12 Unwinding conditions

definition *dynamic-step-consistent* :: *bool* **where**
dynamic-step-consistent $\equiv$ $\forall$ *a d s t. reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\sim$ *d* $\sim$ *t*) $\wedge$
(((*the* (*domain a*)) @ *s* *d*) $\longrightarrow$ (*s* $\sim$ (*the* (*domain a*)) $\sim$ *t*))

$\longrightarrow ((step\ s\ a) \sim d \sim (step\ t\ a))$

definition *dynamic-weakly-step-consistent* :: *bool* **where**
dynamic-weakly-step-consistent $\equiv\ \forall\, a\ d\ s\ t.\ reachable0\ s \wedge reachable0\ t \wedge (s \sim d \sim t) \wedge$
$((the\ (domain\ a))\ @\ s\ \ d) \wedge (s \sim (the\ (domain\ a)) \sim t)$
$\longrightarrow ((step\ s\ a) \sim d \sim (step\ t\ a))$

definition *dynamic-weakly-step-consistent-e* :: *'e* $\Rightarrow$ *bool* **where**
dynamic-weakly-step-consistent-e a $\equiv\ \forall\, d\ s\ t.\ reachable0\ s \wedge reachable0\ t \wedge (s \sim d \sim t) \wedge$
$((the\ (domain\ a))\ @\ s\ \ d) \wedge (s \sim (the\ (domain\ a)) \sim t)$
$\longrightarrow ((step\ s\ a) \sim d \sim (step\ t\ a))$

lemma *dynamic-weakly-step-consistent-all-evt* : *dynamic-weakly-step-consistent* = ($\forall\, a.$ *dynamic-weakly-step-consistent-e a*)
by (*simp add*: *dynamic-weakly-step-consistent-def dynamic-weakly-step-consistent-e-def*)

definition *dynamic-local-respect* :: *bool* **where**
dynamic-local-respect $\equiv \forall\, a\ d\ s.\ reachable0\ s \wedge \neg((the\ (domain\ a))\ @\ s\ \ d) \longrightarrow (s \sim d \sim (step\ s\ a))$

definition *dynamic-local-respect-e* :: *'e* $\Rightarrow$ *bool* **where**
dynamic-local-respect-e a $\equiv \forall\, d\ s.\ reachable0\ s \wedge \neg((the\ (domain\ a))\ @\ s\ \ d) \longrightarrow (s \sim d \sim (step\ s\ a))$

lemma *dynamic-local-respect-all-evt* : *dynamic-local-respect* = ($\forall\, a.$ *dynamic-local-respect-e a*)
by (*simp add*: *dynamic-local-respect-def dynamic-local-respect-e-def*)

declare *dynamic-step-consistent-def* [*cong*] **and** *dynamic-weakly-step-consistent-def* [*cong*] **and**
dynamic-local-respect-def [*cong*]

lemma *step-cons-impl-weak* : *dynamic-step-consistent* $\Longrightarrow$ *dynamic-weakly-step-consistent*
using *dynamic-step-consistent-def dynamic-weakly-step-consistent-def* **by** *blast*

definition *lemma-local* :: *bool* **where**
lemma-local $\equiv \forall\, s\ a\ as\ u.\ the\ (domain\ a) \notin sources\ (a\ \#\ as)\ u\ s \longrightarrow (s \approx (sources\ (a\ \#\ as)\ u\ s)\ \approx (step\ s\ a))$

lemma *weak-with-step-cons*:
assumes *p1*: *dynamic-weakly-step-consistent*
and *p2*: *dynamic-local-respect*
shows *dynamic-step-consistent*
proof –
{
fix *a d s t*
have $reachable0\ s \wedge reachable0\ t \wedge (s \sim d \sim t) \wedge$

$(((the\ (domain\ a))\ @\ s\ \ d) \longrightarrow (s \sim (the\ (domain\ a)) \sim t))$
$\longrightarrow ((step\ s\ a) \sim d \sim (step\ t\ a))$

proof −
{
assume *a0*: *reachable0 s*
assume *a1*: *reachable0 t*
assume *a2*: $(s \sim d \sim t)$
assume *a3*: $(((the\ (domain\ a))\ @\ s\ \ d) \longrightarrow (s \sim (the\ (domain\ a)) \sim t))$
have $((step\ s\ a) \sim d \sim (step\ t\ a))$
proof (*cases* ((*the* (*domain a*)) @ *s d*))
assume *b0*: ((*the* (*domain a*)) @ *s d*)
have *b1*: $(s \sim (the\ (domain\ a)) \sim t)$
using *b0 a3* **by** *auto*
have *b2*: $((step\ s\ a) \sim d \sim (step\ t\ a))$
using *a0 a1 a2 b0 b1 p1 dynamic-weakly-step-consistent-def* **by** *blast*
then show *?thesis* **by** *auto*
next
assume *b0*: ¬((*the* (*domain a*)) @ *s d*)
have *b1*: ¬((*the* (*domain a*)) @ *t d*)
using *a0 a1 a2 b0 policy-respect* **by** *auto*
have *b2*: $s \sim d \sim (step\ s\ a)$
using *b0 p2 a0* **by** *auto*
have *b3*: $t \sim d \sim (step\ t\ a)$
using *b1 p2 a1* **by** *auto*
have *b4*: $((step\ s\ a) \sim d \sim (step\ t\ a))$
using *b2 b3 a2 vpeq-symmetric-lemma vpeq-transitive-lemma* **by**
blast
then show *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed

27.13 Lemmas for the inference framework

lemma *sources-refl*:*reachable0 s* $\Longrightarrow u \in$ *sources as u s*
apply(*induct as arbitrary*: *s*)
apply(*simp add*: *sources-Nil*)
apply(*simp add*: *sources-Cons*)
using *enabled reachableStep*
by *metis*

lemma *lemma-1-sub-1* : ⟦*reachable0 s* ;
dynamic-local-respect;
the (*domain a*) $\notin$ *sources* (*a* # *as*) *u s*;

$(s \approx (sources\ (a\ \#\ as)\ u\ s) \approx t)]\!]$
$\Longrightarrow (s \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ s\ a))$
apply (*simp add*:*dynamic-local-respect-def sources-Cons*)
by *blast*

lemma *lemma-1-sub-2* : [[*reachable0 s* ;
reachable0 t ;
dynamic-local-respect;
the (*domain a*) $\notin$ *sources* (*a* # *as*) *u s*;
$(s \approx (sources\ (a\ \#\ as)\ u\ s) \approx t)]\!]$
$\Longrightarrow (t \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ t\ a))$
proof −
assume *a1*: *reachable0 s*
assume *a2*: *reachable0 t*
assume *a3*: *dynamic-local-respect*
assume *a6*: *the* (*domain a*) $\notin$ *sources* (*a* # *as*) *u s*
assume *a7*: $(s \approx (sources\ (a\ \#\ as)\ u\ s) \approx t)$
have *b1*: $\forall v.\ v \in sources\ as\ u\ (step\ s\ a) \longrightarrow \neg interferes\ (the\ (domain\ a))$
s v
using *a6 sources-Cons* **by** *auto*
have *b2*: *sources* (*a* # *as*) *u s* = *sources as u* (*step s a*)
using *a6 sources-Cons* **by** *auto*
have *b3*: $\forall v.\ v \in sources\ as\ u\ (step\ s\ a) \longrightarrow (s \sim v \sim t)$
using *a7 b2 ivpeq-def* **by** *blast*
have *b4*: $\forall v.\ v \in sources\ as\ u\ (step\ s\ a) \longrightarrow \neg interferes\ (the\ (domain\ a))$
t v
using *a1 a2 policy-respect b1 b3* **by** *blast*
have *b5*: $\forall v.\ v \in sources\ as\ u\ (step\ s\ a) \longrightarrow (t \sim v \sim (step\ t\ a))$
using *a2 a3 b4* **by** *auto*
then show *?thesis*
using *ivpeq-def* **by** *auto*
qed

lemma *lemma-1-sub-3* : [[
the (*domain a*) $\notin$ *sources* (*a* # *as*) *u s*;
$(s \approx (sources\ (a\ \#\ as)\ u\ s) \approx t)]\!]$
$\Longrightarrow (s \approx (sources\ as\ u\ (step\ s\ a)) \approx t)$
apply (*simp add*:*sources-Cons*)
apply (*simp add*:*sources-Cons*)
done

lemma *lemma-1-sub-4* : $[\![(s \approx (sources\ as\ u\ (step\ s\ a)) \approx t)$;
$(s \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ s\ a))$;
$(t \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ t\ a))\ ]\!]$
$\Longrightarrow ((step\ s\ a) \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ t\ a))$
by (*meson ivpeq-def vpeq-symmetric-lemma vpeq-transitive-lemma*)

lemma *lemma-1* : [[*reachable0 s*;

$reachable0\ t;$
$dynamic\text{-}step\text{-}consistent;$
$dynamic\text{-}local\text{-}respect;$
$(s \approx (sources\ (a\ \#\ as)\ u\ s) \approx t)\rrbracket$
$\Longrightarrow ((step\ s\ a) \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ t\ a))$

apply (*case-tac the* (*domain a*)$\in$*sources* (*a* # *as*) *u s*)
apply (*simp add*: *dynamic-step-consistent-def*)
apply (*simp add*: *sources-Cons*)
proof −
assume *a1*: *dynamic-local-respect*
assume *a4*: *the* (*domain a*) $\notin$ *sources* (*a* # *as*) *u s*
assume *a5*: $(s \approx (sources\ (a\ \#\ as)\ u\ s) \approx t)$
assume *b0*: *reachable0 s*
assume *b1*: *reachable0 t*

have *a6*:$(s \approx (sources\ as\ u\ (step\ s\ a)) \approx t)$
using *a1 policy-respect a4 a5 lemma-1-sub-3* **by** *auto*
then have *a7*: $(s \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ s\ a))$
using *b0 a1 policy-respect a4 a5 lemma-1-sub-1* **by** *auto*
then have *a8*: $(t \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ t\ a))$
using *b1 b0 a1 policy-respect a4 a5 lemma-1-sub-2* **by** *auto*
then show $((step\ s\ a) \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ t\ a))$
using *a6 a7 lemma-1-sub-4* **by** *blast*
qed

lemma *lemma-2* : $\llbracket reachable0\ s;$
$dynamic\text{-}local\text{-}respect;$
$the\ (domain\ a) \notin sources\ (a\ \#\ as)\ u\ s\rrbracket$
$\Longrightarrow (s \approx (sources\ as\ u\ (step\ s\ a)) \approx (step\ s\ a))$

apply (*simp add*:*dynamic-local-respect-def*)
apply (*simp add*:*sources-Cons*)
by *blast*

lemma *sources-eq1*: $\forall\, s\ t\ as\ u.\ reachable0\ s\ \wedge$
$reachable0\ t\ \wedge$
$dynamic\text{-}step\text{-}consistent\ \wedge$
$dynamic\text{-}local\text{-}respect\ \wedge$
$(s \approx (sources\ as\ u\ s) \approx t)$
$\longrightarrow (sources\ as\ u\ s) = (sources\ as\ u\ t)$

proof −
{
fix *as*
have $\forall\, s\ t\ u.\ reachable0\ s\ \wedge$
$reachable0\ t\ \wedge$
$dynamic\text{-}step\text{-}consistent\ \wedge$
$dynamic\text{-}local\text{-}respect\ \wedge$
$(s \approx (sources\ as\ u\ s) \approx t)$
$\longrightarrow (sources\ as\ u\ s) = (sources\ as\ u\ t)$
proof(*induct as*)

```
case Nil then show ?case by (simp add: sources-Nil)
next
  case (Cons b bs)
  assume p0: ∀ s t u.((reachable0 s)
                 ∧ (reachable0 t)
                 ∧ dynamic-step-consistent
                 ∧ dynamic-local-respect
                 ∧ (s ≈ (sources bs u s) ≈ t)) ⟶
                   (sources bs u s) = (sources bs u t)
  then show ?case
    proof –
    {
      fix s t u
      assume p1: reachable0 s
      assume p2: reachable0 t
      assume p3: dynamic-step-consistent
      assume p5: dynamic-local-respect
      assume p9: (s ≈ (sources (b # bs) u s) ≈ t)
      have a2: ((step s b) ≈ (sources bs u (step s b)) ≈ (step t b))
        using lemma-1 p1 p2 p3 policy-respect p5 p9 by blast
      have a3: sources (b # bs) u s = sources (b # bs) u t
        proof (cases the (domain b) ∈ (sources (b # bs) u s))
          assume b0: the (domain b) ∈ (sources (b # bs) u s)
          have b1: s ∼ (the(domain b)) ∼ t
            using b0 p9 by auto
         have b3: interferes (the (domain b)) s u = interferes (the (domain
b)) t u
            using p1 p2 policy-respect p9 sources-refl by fastforce
          have b4: (sources bs u (step s b)) = (sources bs u (step t b))
            using a2 p0 p1 p2 p3 p5 reachableStep by blast
          have b5: ∀ v. v∈sources bs u (step s b)
              ⟶ interferes (the (domain b)) s v = interferes (the (domain
b)) t v
         using p1 p2 ivpeq-def policy-respect p9 sources-Cons by fastforce
          then show sources (b # bs) u s = sources (b # bs) u t
            using b4 b5 sources-Cons by auto
        next
          assume b0: the (domain b) ∉ (sources (b # bs) u s)
          have b1: sources (b # bs) u s = sources bs u (step s b)
            using b0 sources-Cons by auto
          have b2: (sources bs u (step s b)) = (sources bs u (step t b))
            using a2 p0 p1 p2 p3 p5 reachableStep by blast
              have b3: ∀ v. v∈sources bs u (step s b)⟶¬ interferes (the
(domain b)) s v
            using b0 sources-Cons by auto
              have b4: ∀ v. v∈sources bs u (step s b)⟶¬ interferes (the
(domain b)) t v
            using b1 b3 p1 p2 p9 policy-respect by fastforce
              have b5: ∀ v. v∈sources bs u (step t b)⟶¬ interferes (the
```

```
(domain b)) t v
              by (simp add: b2 b4)
             have b6: the (domain b) ∉ (sources (b # bs) u t)
              using b0 b2 b5 sources.simps(2) by auto
             have b7: sources (b # bs) u t = sources bs u (step t b)
              using b6 sources-Cons by auto
             then show ?thesis
              by (simp add: b1 b2)
            qed
         }
         then show ?thesis by blast
         qed
      qed

    }
    then show ?thesis by blast
    qed

  lemma ipurge-eq: ∀ s t as u. reachable0 s ∧
             reachable0 t ∧
             dynamic-step-consistent ∧
             dynamic-local-respect ∧
             (s ≈ (sources as u s) ≈ t)
             ⟶ (ipurge as u s) = (ipurge as u t)
    proof −
    {
     fix as
     have ∀ s t u. reachable0 s ∧
             reachable0 t ∧
             dynamic-step-consistent ∧
             dynamic-local-respect ∧
             (s ≈ (sources as u s) ≈ t)
             ⟶ (ipurge as u s) = (ipurge as u t)
      proof(induct as)
       case Nil then show ?case by (simp add: sources-Nil)
      next
       case (Cons b bs)
       assume p0: ∀ s t u.((reachable0 s)
                   ∧ (reachable0 t)
                   ∧ dynamic-step-consistent
                   ∧ dynamic-local-respect
                   ∧ (s ≈ (sources bs u s) ≈ t))
                   ⟶ (ipurge bs u s) = (ipurge bs u t)
       then show ?case
        proof −
        {
          fix s t u
          assume p1: reachable0 s
          assume p2: reachable0 t
```

assume *p3*: *dynamic-step-consistent*
assume *p5*: *dynamic-local-respect*
assume *p9*: $(s \approx (sources\ (b \# bs)\ u\ s) \approx t)$
have *a1*: $((step\ s\ b) \approx (sources\ bs\ u\ (step\ s\ b)) \approx (step\ t\ b))$
  using *lemma-1 p1 p2 p3 p5 p9* **by** *blast*
have *a2*: $(ipurge\ bs\ u\ (step\ s\ b)) = (ipurge\ bs\ u\ (step\ t\ b))$
  using *a1 p0 p1 p2 p3 p5 p9 reachableStep* **by** *blast*
have *a3*: $sources\ (b \# bs)\ u\ s = sources\ (b \# bs)\ u\ t$
  using *p1 p2 p3 p5 p9 sources-eq1* **by** *blast*
have *a4*: $ipurge\ (b \# bs)\ u\ s = ipurge\ (b \# bs)\ u\ t$
  proof $(cases\ the\ (domain\ b) \in (sources\ (b \# bs)\ u\ s))$
    assume *b0*: $the\ (domain\ b) \in (sources\ (b \# bs)\ u\ s)$
    have *b1*: $s \sim (the(domain\ b)) \sim t$
      using *b0 p9* **by** *auto*
    have *b3*: $the\ (domain\ b) \in (sources\ (b \# bs)\ u\ t)$
      using *a3 b0* **by** *auto*
    then show *?thesis*
      using *a2 b0 ipurge-Cons* **by** *auto*
  next
    assume *b0*: $the\ (domain\ b) \notin (sources\ (b \# bs)\ u\ s)$
    have *b1*: $sources\ (b \# bs)\ u\ s = sources\ bs\ u\ (step\ s\ b)$
      using *b0 sources-Cons* **by** *auto*
        have *b3*: $\forall v.\ v \in sources\ bs\ u\ (step\ s\ b) \longrightarrow \neg\ interferes\ (the$
$(domain\ b))\ s\ v$
      using *b0 sources-Cons* **by** *auto*
        have *b4*: $\forall v.\ v \in sources\ bs\ u\ (step\ s\ b) \longrightarrow \neg\ interferes\ (the$
$(domain\ b))\ t\ v$
      using *b1 b3 p1 p2 p9 policy-respect* **by** *fastforce*
    have *b5*: $the\ (domain\ b) \notin (sources\ (b \# bs)\ u\ t)$
      using *a3 b1 b4 interf-reflexive* **by** *auto*
    have *b6*: $ipurge\ (b \# bs)\ u\ s = ipurge\ bs\ u\ (step\ s\ b)$
      using *b0* **by** *auto*
    have *b7*: $ipurge\ (b \# bs)\ u\ t = ipurge\ bs\ u\ (step\ t\ b)$
      using *b5* **by** *auto*
    then show *?thesis*
      using *b6 b7 a2* **by** *auto*
  qed
}
then show *?thesis* **by** *blast*
qed
qed
}
then show *?thesis* **by** *blast*
qed

lemma *non-influgence-lemma*: $\forall s\ t\ as\ u.\ reachable0\ s \wedge$
  $reachable0\ t \wedge$
  $dynamic\text{-}step\text{-}consistent \wedge$
  $dynamic\text{-}local\text{-}respect \wedge$

$(s \approx (sources\ as\ u\ s) \approx t)$
$\longrightarrow ((s\ \ as \cong t\ \ (ipurge\ as\ u\ t)\ @\ u))$
proof –
{
fix *as*
have $\forall\, s\ t\ u.\ reachable0\ s\ \wedge$
$reachable0\ t\ \wedge$
$dynamic\text{-}step\text{-}consistent\ \wedge$
$dynamic\text{-}local\text{-}respect\ \wedge$
$(s \approx (sources\ as\ u\ s) \approx t)$
$\longrightarrow ((s\ \ as \cong t\ \ (ipurge\ as\ u\ t)\ @\ u))$
proof (*induct as*)
case *Nil* **show** *?case* **using** *sources-Nil* **by** *auto*
next
case (*Cons b bs*)
assume *p0*: $\forall\, s\ t\ u.((reachable0\ s)$
$\wedge\ (reachable0\ t)$
$\wedge\ dynamic\text{-}step\text{-}consistent$
$\wedge\ dynamic\text{-}local\text{-}respect$
$\wedge\ (s \approx (sources\ bs\ u\ s) \approx t)) \longrightarrow$
$((s\ \ bs \cong t\ \ (ipurge\ bs\ u\ t)\ @\ u))$
then show *?case*
proof –
{
fix *s t u*
assume *p1*: *reachable0 s*
assume *p2*: *reachable0 t*
assume *p3*: *dynamic-step-consistent*
assume *p4*: *dynamic-local-respect*
assume *p8*: $(s \approx (sources\ (b\ \#\ bs)\ u\ s) \approx t)$
have *a1*: $((step\ s\ b) \approx (sources\ bs\ u\ (step\ s\ b)) \approx (step\ t\ b))$
using *lemma-1 p1 p2 p3 p4 p8* **by** *blast*
have $s\ \ b\ \#\ bs \cong t\ \ ipurge\ (b\ \#\ bs)\ u\ t\ @\ u$
proof (*cases the* (*domain b*) $\in$ *sources* (b # *bs*) *u s*)
assume *b0*: *the* (*domain b*) $\in$ *sources* (b # *bs*) *u s*
have *b1*: *interferes* (*the* (*domain b*)) *s u* = *interferes* (*the* (*domain b*)) *t u*
using *p1 p2 policy-respect p8 sources-refl* **by** *fastforce*
have *b2*: $\forall\, v.\ v{\in}sources\ bs\ u\ (step\ s\ b)$
$\longrightarrow$ *interferes* (*the* (*domain b*)) *s v* = *interferes* (*the* (*domain b*)) *t v*
using *p1 p2 ivpeq-def policy-respect p8 sources-Cons* **by** *fastforce*
have *b3*: *ipurge* (b # *bs*) *u t* = b # (*ipurge bs u* (*step t b*))
by (*metis b0 ipurge-Cons p1 p2 p3 p4 p8 sources-eq1*)
have *b4*: $(((step\ s\ b)\ \ bs \cong (step\ t\ b)\ \ (ipurge\ bs\ u\ (step\ t\ b))\ @\ u))$
using *a1 p0 p1 p2 p3 p4 reachableStep* **by** *blast*
show *?thesis*
using *b3 b4* **by** *auto*
next

assume *b0*: *the* (*domain* *b*) ∉ *sources* (*b* # *bs*) *u* *s*
have *b1*: *ipurge* (*b* # *bs*) *u* *t* = (*ipurge* *bs* *u* (*step* *t* *b*))
by (*metis a1 b0 ipurge-Cons ipurge-eq p1 p2 p3 p4 p8 reachableStep*)
have *b2*: (*s* ≈ (*sources* *bs* *u* (*step* *s* *b*)) ≈ (*step* *s* *b*))
using *b0 lemma-2 p1 p4* **by** *blast*
have *b3*:(*s* ≈ (*sources* *bs* *u* (*step* *s* *b*)) ≈ *t*)
using *b0 lemma-1-sub-3 p8* **by** *blast*
have *b4*: ((*step* *s* *b*) ≈ (*sources* *bs* *u* (*step* *s* *b*)) ≈ *t*)
by (*meson b3 b2 ivpeq-def vpeq-symmetric-lemma vpeq-transitive-lemma*)
have *b5*: (((*step* *s* *b*) *bs* ≅ *t* (*ipurge* *bs* *u* *t*) @ *u*))
using *b4 p0 p1 p2 p3 p4 reachableStep* **by** *blast*
have *b6*: (*t* ≈ (*sources* *bs* *u* (*step* *s* *b*)) ≈ (*step* *t* *b*))
using *p1 p2 b0 lemma-1-sub-2 p4 p8* **by** *blast*
have *b7*: *ipurge* *bs* *u* *t* = *ipurge* *bs* *u* (*step* *t* *b*)
by (*metis a1 b4 ipurge-eq p1 p2 p3 p4 reachableStep*)
have *b8*: (((*step* *s* *b*) *bs* ≅ *t* (*ipurge* *bs* *u* (*step* *t* *b*)) @ *u*))
using *b5 b7* **by** *auto*
then show *?thesis*
using *b1 observ-equivalence-def run-Cons* **by** *auto*
qed
}
then show *?thesis* **by** *blast*
qed
qed
}
then show *?thesis* **by** *blast*
qed

27.14 Interference framework of information flow security properties

theorem *nonintf-impl-weak*: *noninterference* ⟹ *weak-noninterference*
by (*metis noninterference-def observ-equiv-sym observ-equiv-trans reachable-s0 weak-noninterference-def*)

theorem *wk-nonintf-r-impl-wk-nonintf*: *weak-noninterference-r* ⟹ *weak-noninterference*
using *reachable-s0* **by** *auto*

theorem *nonintf-r-impl-noninterf*: *noninterference-r* ⟹ *noninterference*
using *noninterference-def noninterference-r-def reachable-s0* **by** *auto*

theorem *nonintf-r-impl-wk-nonintf-r*: *noninterference-r* ⟹ *weak-noninterference-r*
by (*metis noninterference-r-def observ-equiv-sym observ-equiv-trans weak-noninterference-r-def*)

lemma *noninf-impl-nonintf-r*: *noninfluence* ⟹ *noninterference-r*
using *ivpeq-def noninfluence-def noninterference-r-def vpeq-reflexive-lemma*
by *blast*

lemma *noninf-impl-nonlk*: *noninfluence* ⟹ *nonleakage*

using *noninterference-r-def nonleakage-def observ-equiv-sym observ-equiv-trans noninfluence-def noninf-impl-nonintf-r* **by** *blast*

lemma *wk-noninfl-impl-nonlk*: *weak-noninfluence* $\Longrightarrow$ *nonleakage*
using *weak-noninfluence-def nonleakage-def* **by** *blast*

lemma *wk-noninfl-impl-wk-nonintf-r*: *weak-noninfluence* $\Longrightarrow$ *weak-noninterference-r*
using *ivpeq-def weak-noninfluence-def vpeq-reflexive-lemma weak-noninterference-r-def* **by** *blast*

lemma *sources-step2*:
$[\![reachable0\ s;\ (the\ (domain\ a))@s\ \ d]\!] \Longrightarrow sources\ [a]\ d\ s = \{the\ (domain\ a),d\}$
apply(*auto simp*: *sources-Cons sources-Nil enabled dest*: *enabled*)
done

lemma *exec-equiv-both*:
$[\![reachable0\ C1;\ reachable0\ C2;(step\ C1\ a)\ \ as \cong (step\ C2\ b)\ \ bs\ @\ u]\!]$
$\Longrightarrow (C1\ \ (a\ \#\ as) \cong C2\ \ (b\ \#\ bs)\ @\ u)$
by *auto*

lemma *sources-unwinding-step*:
$[\![reachable0\ s;\ reachable0\ t;s \approx(sources\ (a\#as)\ d\ s)\approx\ t;\ dynamic\text{-}step\text{-}consistent]\!]$

$\Longrightarrow ((step\ s\ a) \approx(sources\ as\ d\ (step\ s\ a))\approx\ (step\ t\ a))$
apply(*clarsimp simp*: *ivpeq-def sources-Cons*)
using *UnionI dynamic-step-consistent-def* **by** *blast*

lemma *nonlk-imp-sc*: *nonleakage* $\Longrightarrow$ *dynamic-step-consistent*
proof −
assume *p0*: *nonleakage*
have *p1*: $\forall\ as\ d\ s\ t.\ reachable0\ s \wedge reachable0\ t$
$\wedge\ (s \approx (sources\ as\ d\ s) \approx t) \longrightarrow (s\ \ as \cong t\ \ as\ @\ d)$
using *p0 nonleakage-def* **by** *auto*
have *p2*: $\forall\ a\ d\ s\ t.\ reachable0\ s \wedge reachable0\ t \wedge (s \sim d \sim t) \wedge$
$(((the\ (domain\ a))\ @\ s\ \ d) \longrightarrow (s \sim (the\ (domain\ a)) \sim t))$
$\longrightarrow ((step\ s\ a) \sim d \sim (step\ t\ a))$
proof −
{
fix *a d s t*
assume *a0*: $reachable0\ s \wedge reachable0\ t \wedge (s \sim d \sim t) \wedge$
$(((the\ (domain\ a))\ @\ s\ \ d) \longrightarrow (s \sim (the\ (domain\ a)) \sim t))$
have *a4*: $s \approx (sources\ []\ d\ s) \approx t$
using *a0 sources-Nil* **by** *auto*
have *a5*: $(s\ \ [] \cong t\ \ []\ @\ d)$
using *a4 a0 p1* **by** *auto*
have *a6*: $((step\ s\ a) \sim d \sim (step\ t\ a))$
proof (*cases* $(the\ (domain\ a))@s\ \ d$)
assume *b0*: $(the\ (domain\ a))@s\ \ d$

```
        have b1: sources [a] d s = {d, (the(domain a))}
          using b0 sources-Cons sources-Nil by auto
        have c0: (s ∼ (the (domain a)) ∼ t)
          using b0 a0 by auto
        have b2: s ≈ (sources [a] d s) ≈ t
          using b1 a0 c0 by auto
        have b3: (s  [a] ≅ t  [a] @ d)
          using b2 a0 p1 by auto
        have b4: ((step s a) ∼ d ∼ (step t a))
          using b3 by auto
        then show ?thesis by auto
      next
        assume b0: ¬((the (domain a))@s  d)
        have b1: sources [a] d s = {d}
          using b0 sources-Cons sources-Nil by auto
        have b2: (s ≈ (sources [a] d s) ≈ t)
          using b1 a0 by auto
        have b3: (s  [a] ≅ t  [a] @ d)
          using b2 a0 p1 by auto
        have b4: ((step s a) ∼ d ∼ (step t a))
          using b3 by auto
        then show ?thesis by auto
      qed
      }
    then show ?thesis
      by auto
  qed
 then show ?thesis by auto
qed

lemma sc-imp-nonlk: dynamic-step-consistent ⟹ nonleakage
 proof −
  assume p0: dynamic-step-consistent
  have p1: ∀ a d s t. reachable0 s ∧ reachable0 t ∧ (s ∼ d ∼ t) ∧
       (s ∼ (the (domain a)) ∼ t) ⟶ ((step s a) ∼ d ∼ (step t a))
    using p0 dynamic-step-consistent-def by auto
  have p2: ∀ as d s t. reachable0 s ∧ reachable0 t
         ∧ (s ≈ (sources as d s) ≈ t) ⟶ (s  as ≅ t  as @ d)
    proof −
    {
      fix as
      have ∀ d s t. reachable0 s ∧ reachable0 t
         ∧ (s ≈ (sources as d s) ≈ t) ⟶ (s  as ≅ t  as @ d)
        proof (induct as)
          case Nil show ?case using sources-refl by auto
        next
          case (Cons b bs)
          assume a0: ∀ d s t. reachable0 s ∧ reachable0 t
           ∧ (s ≈ (sources bs d s) ≈ t) ⟶ (s  bs ≅ t  bs @ d)
```

```
          show ?case
            proof -
            {
              fix d s t
              assume b0: reachable0 s ∧ reachable0 t
              assume b1: (s ≈ (sources (b#bs) d s) ≈ t)
              have b2: ((step s b) ≈(sources bs d (step s b))≈ (step t b))
                using b0 b1 p0 sources-unwinding-step by auto
              have b3: (step s b)  bs ≅ (step t b)  bs @ d
                using Cons.hyps b0 b2 reachableStep by blast
              have b4: s  b # bs ≅ t  b # bs @ d
                using b3 by auto
              }
              then show ?thesis by auto
          qed
        qed
      }
    then show ?thesis by auto
  qed
 then show ?thesis by auto
qed

theorem sc-eq-nonlk: dynamic-step-consistent = nonleakage
 using nonlk-imp-sc sc-imp-nonlk by blast

lemma noninf-imp-dlr: noninfluence ⟹ dynamic-local-respect
 proof -
  assume p0: noninfluence
  have p1: ∀ d as s t. reachable0 s ∧ reachable0 t
          ∧ (s ≈ (sources as d s) ≈ t)
          ⟶ (s  as ≅ t  (ipurge as d t) @ d)
    using p0 noninfluence-def by auto
  have ∀ a d s. reachable0 s ∧ ¬((the (domain a)) @ s  d)
              ⟶ (s ∼ d ∼ (step s a))
    proof -
    {
      fix a d s
      assume a0: reachable0 s ∧ ¬((the (domain a)) @ s  d)
      have a1: sources [a] d s = {d}
        using a0 sources-Cons sources-Nil by auto
      have a2: (ipurge [a] d s) = []
        using a0 a1 interf-reflexive by auto
      have a3: s ∼ d ∼ s
        using vpeq-reflexive-lemma by auto
      have a4: (s ≈ (sources [a] d s) ≈ s)
        using a1 a3 by auto
      have a5: (s  [a] ≅ s  (ipurge [a] d s) @ d)
        using a4 a0 p1 by auto
      have a6: (s  [a] ≅ s  [] @ d)
```

```
        using a5 a2 by auto
      have a7: (s ∼ d ∼ (step s a))
        using a6 vpeq-symmetric-lemma by auto
      }
    then show ?thesis by auto
  qed
 then show ?thesis by auto
qed

lemma noninf-imp-sc: noninfluence ⟹ dynamic-step-consistent
 using nonlk-imp-sc noninf-impl-nonlk by blast

theorem UnwindingTheorem : ⟦dynamic-step-consistent;
                     dynamic-local-respect⟧
                     ⟹ noninfluence
 proof −
  assume p3: dynamic-step-consistent
  assume p4: dynamic-local-respect
  {
  fix as d
  have ∀ s t. reachable0 s ∧
          reachable0 t ∧
          (s ≈ (sources as d s) ≈ t)
          ⟶ ((s  as ≅ t  (ipurge as d  t) @ d))
    proof(induct as)
     case Nil show ?case using sources-Nil by auto
    next
     case (Cons b bs)
     assume p0: ∀ s t. reachable0 s ∧
          reachable0 t ∧
          (s ≈ (sources bs d s) ≈ t)
          ⟶ ((s  bs ≅ t  (ipurge bs d  t) @ d))
     then show ?case
      proof −
      {
       fix s t
       assume p1: reachable0 s
       assume p2: reachable0 t
       assume p8: (s ≈ (sources (b # bs) d s) ≈ t)
       have a1: ((step s b) ≈ (sources bs d (step s b)) ≈ (step t b))
        using lemma-1 p1 p2 p3 p4 p8 by blast
       have a2: s  b # bs ≅ t  ipurge (b # bs) d t @ d
        proof (cases the (domain b) ∈ sources (b # bs) d s)
         assume b0: the (domain b) ∈ sources (b # bs) d s
         have b1: interferes (the (domain b)) s d = interferes (the (domain
b)) t d
          using p1 p2 policy-respect p8 sources-refl by fastforce
         have b2: ∀ v. v∈sources bs d (step s b)
            ⟶ interferes (the (domain b)) s v = interferes (the (domain
```

b)) t v
using p1 p2 ivpeq-def policy-respect p8 sources-Cons **by** fastforce
have b3: ipurge (b # bs) d t = b # (ipurge bs d (step t b))
by (metis b0 ipurge-Cons p1 p2 p3 p4 p8 sources-eq1)
have b4: (((step s b) bs ≅ (step t b) (ipurge bs d (step t b)) @ d))
using a1 p0 p1 p2 p3 p4 reachableStep **by** blast
then show ?thesis
using b3 b4 **by** auto
next
assume b0: the (domain b) ∉ sources (b # bs) d s
have b1: ipurge (b # bs) d t = (ipurge bs d (step t b))
by (metis a1 b0 ipurge-Cons ipurge-eq p1 p2 p3 p4 p8 reachableStep)
have b2: (s ≈ (sources bs d (step s b)) ≈ (step s b))
using b0 lemma-2 p1 p4 **by** blast
have b3:(s ≈ (sources bs d (step s b)) ≈ t)
using b0 lemma-1-sub-3 p8 **by** blast
have b4: ((step s b) ≈ (sources bs d (step s b)) ≈ t)
by (meson b3 b2 ivpeq-def vpeq-symmetric-lemma vpeq-transitive-lemma)
have b5: (((step s b) bs ≅ t (ipurge bs d t) @ d))
using b4 p0 p1 p2 p3 p4 reachableStep **by** blast
have b6: (t ≈ (sources bs d (step s b)) ≈ (step t b))
using p1 p2 b0 lemma-1-sub-2 p4 p8 **by** blast
have b7: ipurge bs d t = ipurge bs d (step t b)
by (metis a1 b4 ipurge-eq p1 p2 p3 p4 reachableStep)
have b8: (((step s b) bs ≅ t (ipurge bs d (step t b)) @ d))
using b5 b7 **by** auto
then show ?thesis
using b1 observ-equivalence-def run-Cons **by** auto
qed
}
then show ?thesis **by** blast
qed
qed
}
then show ?thesis **using** noninfluence-def **by** blast
qed

theorem UnwindingTheorem1 : ⟦dynamic-weakly-step-consistent;
dynamic-local-respect⟧ ⟹ noninfluence
using UnwindingTheorem weak-with-step-cons **by** blast

theorem uc-eq-noninf : (dynamic-step-consistent ∧ dynamic-local-respect) = noninfluence
using UnwindingTheorem1 step-cons-impl-weak noninf-imp-dlr noninf-imp-sc **by** blast

theorem noninf-impl-weak:noninfluence ⟹ weak-noninfluence

proof −
assume *p0*: *noninfluence*
have *p1*: $\forall$ *d as s t. reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\longrightarrow$ (*s as* $\cong$ *t* (*ipurge as d t*) @ *d*)
using *p0 noninfluence-def* **by** *auto*
have *p2*: (*dynamic-step-consistent* $\wedge$ *dynamic-local-respect*)
using *p0 uc-eq-noninf* **by** *auto*
have $\forall$ *d as bs s t . reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\wedge$ *ipurge as d t* = *ipurge bs d t*
$\longrightarrow$ (*s as* $\cong$ *t bs* @ *d*)
proof −
{
fix *d as bs s t*
assume *a0*: *reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\wedge$ *ipurge as d t* = *ipurge bs d t*
have *a4*: *noninterference-r*
using *noninf-impl-nonintf-r p0* **by** *auto*
have *a7*: *weak-noninterference-r*
using *a4 nonintf-r-impl-wk-nonintf-r* **by** *auto*
have *a6*: *ipurge as d s* = *ipurge as d t*
using *a0 p2 ipurge-eq* **by** *auto*
have *b1*: (*s as* $\cong$ *t* (*ipurge as d t*) @ *d*)
using *a0 p1* **by** *auto*
have *b4*: (*s as* $\cong$ *t as* @ *d*)
using *a0 noninf-imp-sc nonleakage-def p0 sc-imp-nonlk* **by** *blast*
have *b5*: (*t bs* $\cong$ *t* (*ipurge bs d t*) @ *d*)
using *a0 a4* **by** *auto*
have *b6*: (*t bs* $\cong$ *t* (*ipurge as d t*) @ *d*)
using *b5 a0* **by** *auto*
have *b7*: (*s as* $\cong$ *t bs* @ *d*)
using *a0 b1 b6 observ-equiv-sym observ-equiv-trans* **by** *blast*
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed

lemma *wk-nonintf-r-and-nonlk-impl-noninfl*: $[\![$*weak-noninterference-r*; *nonleakage*$]\!] \Longrightarrow$ *weak-noninfluence*
proof −
assume *p0*: *weak-noninterference-r*
assume *p1*: *nonleakage*
have *p2*: $\forall$ *d as bs s. reachable0 s* $\wedge$ *ipurge as d s* = *ipurge bs d s*
$\longrightarrow$ (*s as* $\cong$ *s bs* @ *d*)
using *weak-noninterference-r-def p0* **by** *auto*
have *p3*: $\forall$ *d as s t. reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*) $\longrightarrow$ (*s as* $\cong$ *t as* @ *d*)

using *nonleakage-def p1* **by** *auto*
have $\forall$ *d as bs s t* . *reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\wedge$ *ipurge as d t* = *ipurge bs d t*
$\longrightarrow$ (*s as* $\cong$ *t bs* @ *d*)
proof –
{
fix *d as bs s t*
assume *a0*: *reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\wedge$ *ipurge as d t* = *ipurge bs d t*
have *a1*: *s as* $\cong$ *t as* @ *d*
using *a0 p3* **by** *blast*
have *a2*: *t as* $\cong$ *t bs* @ *d*
using *a0 p2* **by** *auto*
have *a3*: (*s as* $\cong$ *t bs* @ *d*)
using *a0 a1 a2 observ-equiv-trans* **by** *blast*
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed

lemma *nonintf-r-and-nonlk-impl-noninfl*: ⟦*noninterference-r*; *nonleakage*⟧ $\Longrightarrow$ *noninfluence*
proof –
assume *p0*: *noninterference-r*
assume *p1*: *nonleakage*
have *p2*: $\forall$ *d as s*. *reachable0 s* $\longrightarrow$ (*s as* $\cong$ *s* (*ipurge as d s*) @ *d*)
using *p0 noninterference-r-def* **by** *auto*
have *p3*: $\forall$ *d as s t*. *reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*) $\longrightarrow$ (*s as* $\cong$ *t as* @ *d*)
using *p1 nonleakage-def* **by** *auto*
have $\forall$ *d as s t*. *reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\longrightarrow$ (*s as* $\cong$ *t* (*ipurge as d t*) @ *d*)
proof –
{
fix *d as bs s t*
assume *a0*: *reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
have *a1*: *s as* $\cong$ *t as* @ *d*
using *p3 a0* **by** *blast*
have *a2*: *s as* $\cong$ *s* (*ipurge as d s*) @ *d*
using *a0 p2* **by** *fast*
have *a3*: *t as* $\cong$ *t* (*ipurge as d t*) @ *d*
using *a0 p2* **by** *fast*
have *s as* $\cong$ *t* (*ipurge as d t*) @ *d*
using *a0 a1 a3 observ-equiv-trans* **by** *blast*
}

then show *?thesis* **by** *auto*
qed
then show *?thesis* **using** *noninfluence-def* **by** *blast*
qed

theorem *nonintf-r-and-nonlk-eq-strnoninfl*: (*noninterference-r* $\wedge$ *nonleakage*) = *noninfluence*
using *nonintf-r-and-nonlk-impl-noninfl noninf-impl-nonintf-r noninf-impl-nonlk* **by** *blast*

end

end

28 Security policy model of Linux Security Module

theory
LSM-SPM
imports
Dynamic-model
begin

locale *LSM-Security-model* = *SM-enabled s0 step domain vpeq interferes*
for *s0* :: $'s$ **and**
step :: $'s \Rightarrow 'e \Rightarrow 's$ **and**
domain :: $'e \Rightarrow ('d$ *option*) **and**
vpeq :: $'s \Rightarrow 'd \Rightarrow 's \Rightarrow$ *bool* ((- $\sim$ - $\sim$ -)) **and**
interferes :: $'d \Rightarrow 's \Rightarrow 'd \Rightarrow$ *bool* ((- @ - -))
+

fixes *observe* :: $'s \Rightarrow 'd \Rightarrow 'obj$ *set* (**infixl** *65*)
and *alter* :: $'s \Rightarrow 'd \Rightarrow 'obj$ *set* (**infixl** *66*)
and *contents* :: $'s \Rightarrow 'obj \Rightarrow 'v$

assumes *contents-consistent*: ($\forall$ *s u t*. (*s* $\sim$ *u* $\sim$ *t*) $\longrightarrow$($\forall$ *n* $\in$ *observe s u*. *contents s n* = *contents t n*))
and *observed-consistent* : ($\forall$ *s t u*. ((*s* $\sim$ *u* $\sim$ *t*) $\longrightarrow$ *s u* = *t u*))
and *ac-interferes* : $\forall$ *s u v*. (*alter s u* $\cap$ *observe s v*) $\neq$ {} $\longrightarrow$(*u* @ *s v*)
begin

definition *drma2s* :: *bool*
where *drma2s* $\equiv$ ($\forall$ *s t a n*. (*n* $\in$ *alter s* (*the*(*domain a*))$\cap$ *alter t* (*the*(*domain a*))) $\wedge$
(*s* $\sim$ (*the* (*domain a*)) $\sim$ *t*) $\wedge$
(*contents s n* = *contents t n*)

$\longrightarrow$ (*contents* (*step s a*) *n* = *contents* (*step t a*) *n*))

definition *drma2* :: *bool*
where *drma2* $\equiv$ ($\forall$ *s t a n*.((*s* $\sim$ *the*(*domain a*) $\sim$ *t*) $\wedge$
((*contents* (*step s a*) *n*) $\neq$ (*contents s n*)
$\vee$(*contents* (*step t a*) *n*) $\neq$ (*contents t n*)))
$\longrightarrow$ (*contents* (*step s a*) *n* = *contents* (*step t a*) *n*))

definition *drma3s* :: *bool*
where*drma3s* $\equiv$ ($\forall$ *a n s* .(*contents* (*step s a*) *n*) $\neq$ (*contents s n*)
$\longrightarrow$ *n* $\in$ *alter s* (*the*(*domain a*)) $\wedge$ *n* $\in$ *alter* (*step s a*) (*the*(*domain a*)))

definition *drma4s* :: *bool*
where *drma4s* $\equiv$ ($\forall$ *s u a*. (((*step s a*) *u*) $-$ (*s* *u*)) $\subseteq$ (*s* (*the*(*domain a*))))

definition *drma5s* :: *bool*
where *drma5s* $\equiv$ ($\forall$ *s t u v*. (*u* @ *s* *v*) $\wedge$ (*u* @ *t* *v*))

definition *drma5s'* $\equiv$ $\forall$ *s t u v*. (*s* $\sim$ *u* $\sim$ *t*) $\wedge$ (*s* $\sim$ *v* $\sim$ *t*) $\longrightarrow$ (*alter s v* $\cap$ *observe s u*) = (*alter t v* $\cap$ *observe t u*)

definition *drma3* $\equiv$ ($\forall$ *a n s* .(*contents* (*step s a*) *n*) $\neq$ (*contents s n*)
$\longrightarrow$ *n* $\in$ *alter s* (*the*(*domain a*)))

end
end

29 The Core of the linux kernel abstract event

theory
kernelA
imports
Linux-LSM-Model
LSM-SPM
begin

29.1 kernel

locale *Kernel* = *lsm* +
fixes *k-superblock* :: *'a* $\Rightarrow$*t-sb* $\rightharpoonup$*super-block*
fixes *sdentry* :: *'a* $\Rightarrow$ *dname*$\rightharpoonup$ *dentry*
fixes *sockets* :: *'a* $\Rightarrow$*socketdesp* $\rightharpoonup$ *socket*
fixes *keys* :: *'a* $\Rightarrow$*keyid* $\rightharpoonup$ *key*
fixes *kfiles* :: *'a* $\Rightarrow$*fname* $\rightharpoonup$ *Files*
fixes *msg-msgs* ::*'a* $\Rightarrow$*msg-mid* $\rightharpoonup$ *msg-msg*
fixes *msg-queues* :: *'a* $\Rightarrow$*msg-qid* $\rightharpoonup$ *kern-ipc-perm*
fixes *contents* :: *'a* $\Rightarrow$ *Obj* $\Rightarrow$ *'v*

begin
definition *current-process* :: $'a \Rightarrow$ *Task*
where *current-process s* = *the* (*k-task s* (*current s*))

definition *current-cred* :: $'a \Rightarrow$ *Cred*
where *current-cred s* = *cred* (*current-process s*)

definition *current-real-cred* :: $'a \Rightarrow$ *Cred*
where *current-real-cred s* = *real-cred* (*current-process s*)

definition *task-cred* :: $'a \Rightarrow$ *Task* $\Rightarrow$*Cred*
where *task-cred s t* = *real-cred t*

definition *get-process-by-pid* :: $'a \Rightarrow$ *nat* $\Rightarrow$ *Task*
where*get-process-by-pid s p* $\equiv$ *the*((*k-task s*) *p*)

definition *get-processid* :: $'a \Rightarrow$ *Task* $\Rightarrow$ *nat*
where*get-processid s t* $\equiv$ *SOME x* . (*k-task s*) *x* = *Some t*

definition *get-inode* :: $'a \Rightarrow$ *inum* $\Rightarrow$ *inode*
where *get-inode s inum* $\equiv$ *SOME x* .(*inodes s*) *inum* = *Some x*

definition *get-dentry* :: $'a \Rightarrow$ *string* $\Rightarrow$ *dentry*
where *get-dentry s dname* $\equiv$ *SOME x* .(*sdentry s*) *dname* = *Some x*

definition *get-file* :: $'a \Rightarrow$*string* $\Rightarrow$ *Files*
where *get-file s name* $\equiv$ *SOME x* .(*kfiles s*) *name* = *Some x*

definition *get-socket* :: $'a \Rightarrow$ *socketdesp* $\Rightarrow$ *socket*
where *get-socket s dsp* $\equiv$ *SOME x* .(*sockets s*) *dsp* = *Some x*

definition *current-sbs* :: $'a \Rightarrow$ *t-sb set*
where *current-sbs s* = {*t* .$\forall$ *sb* .*k-superblock s* = (*k-superblock s*) (*t* := *Some sb*) }

definition *current-tasks* :: $'a \Rightarrow$ *process-id set*
where *current-tasks s* = {*t* .$\forall$ *sb* .*k-task s* = (*k-task s*) (*t* := *Some sb*) }

datatype *Event-sb* =
Event-sb-copy-data process-id t-sb
| *Event-sb-remount process-id path Void*
| *Event-sb-kern-mount process-id file-system-type int string string*
| *Event-sb-show-options process-id seq-file t-sb*
| *Event-sb-statfs process-id dentry*
| *Event-sb-mount process-id string string string nat Void*
| *Event-sb-umount process-id mount int*
| *Event-sb-pivotroot process-id*

| *Event-set-mnt-opts process-id btrfs-fs-info super-block opts*
| *Event-set-sb-security process-id super-block dentry nfs-mount-info*
| *Event-sb-clone-mnt-opts process-id super-block dentry nfs-mount-info*
| *Event-sb-parse-opts-str process-id string opts*

datatype *Event-tsk* =
Event-prepare-creds process-id

| *Event-sys-setreuid process-id kuid kuid*
| *Event-setpgid process-id pid-t pid-t*
| *Event-do-getpgid process-id pid-t*
| *Event-getsid process-id pid-t*
| *Event-getsecid process-id Task u32*
| *Event-task-setnice process-id Task int*
| *Event-set-task-ioprio process-id Task int*
| *Event-get-task-ioprio process-id Task*
| *Event-check-prlimit-permission process-id Task nat*
| *Event-do-prlimit process-id Task nat*
| *Event-task-setscheduler process-id Task*
| *Event-task-getscheduler process-id Task*
| *Event-task-movememory process-id Task*
| *Event-task-kill process-id Task siginfo int Cred*
| *Event-task-prctl process-id int nat nat nat nat*

datatype *Event-Binder* = *Event-binder-set-context-mgr process-id Task*
| *Event-binder-transaction process-id Task Task*
| *Event-binder-transfer-binder process-id Task Task*
| *Event-binder-transfer-file process-id Task Task*

datatype *Event-Ptrace* = *Event-ptrace-access-check process-id Task nat*
| *Event-ptrace-traceme process-id*

datatype *Event-sys* = *Event-smack-syslog process-id int*
| *Event-prepare-binprm process-id linux-binprm*

datatype *Event-ipc* = *Event-ipc-permission process-id kern-ipc-perm nat*
| *Event-ipc-getsecid process-id kern-ipc-perm*
| *Event-msg-queue-associate process-id kern-ipc-perm int*
| *Event-msg-queue-msgctl process-id kern-ipc-perm IPC-CMD*
| *Event-msg-queue-msgsnd process-id kern-ipc-perm msg-msg int*
| *Event-msg-queue-msgrcv process-id kern-ipc-perm msg-msg Task int int*
| *Event-shm-associate process-id kern-ipc-perm int*
| *Event-shm-shmctl process-id kern-ipc-perm IPC-CMD*
| *Event-shm-shmat process-id kern-ipc-perm string int*
| *Event-sem-associate process-id kern-ipc-perm int*
| *Event-sem-semctl process-id kern-ipc-perm IPC-CMD*

| *Event-sem-semop process-id kern-ipc-perm sembuf nat int*

datatype *Event-inode* =
Event-vfs-link process-id dentry inode dentry inode
| *Event-vfs-unlink process-id inode dentry inode*
| *Event-vfs-rmdir process-id inode dentry*
| *Event-vfs-rename process-id inode dentry inode dentry inode nat*
| *Event-inode-permission process-id inode int*
| *Event-notify-change process-id dentry iattr inode*
| *Event-fat-ioctl-set-attributes process-id Files*
| *Event-vfs-getattr process-id path*
| *Event-vfs-setxattr process-id dentry xattr string nat nat*

| *Event-vfs-getxattr process-id dentry xattr string nat*
| *Event-vfs-removexattr process-id dentry xattr*
| *Event-xattr-getsecurity process-id inode xattr string int*

| *Event-nfs4-listxattr-nfs4-label process-id inode string int*
| *Event-sockfs-listxattr process-id dentry string int*

datatype *Event-network-sock* = *Event-unix-stream-connect process-id socket sockaddr int int*
| *Event-unix-dgram-connect process-id socket sockaddr int int*
| *Event-unix-dgram-sendmsg process-id socket sockaddr int*

| *Event-sys-bind′ process-id int sockaddr int*
| *Event-sys-connect′ process-id int sockaddr int*
| *Event-sock-sendmsg process-id socket msghdr*
| *Event-sock-recvmsg process-id socket msghdr int*
| *Event-sk-filter-trim-cap process-id sock sk-buff int*
| *Event-sock-getsockopt process-id socket int int string int*
| *Event-unix-get-peersec-dgram process-id socket scm-cookie*

datatype *Event-network* = *Event-sk-alloc*
| *Event-sk-classify-flow*
| *Event-req-classify-flow*
| *Event-inet-conn-established*
| *Event-secmark-relabel-packet*
| *Event-secmark-refcount-inc*
| *Event-secmark-refcount-dec*
| *Event-tun-dev-alloc-Event*
| *Event-tun-dev-free-Event*
| *Event-tun-dev-create*
| *Event-tun-dev-attach-queue*
| *Event-tun-dev-attach*
| *Event-tun-dev-open*
| *Event-sctp-assoc-request*

| *Event-sctp-bind-connect*
| *Event-sctp-sk-clone*

datatype *Event-Infiniband* = *Event-ib-pkey-access*
| *Event-ib-endport-manage-subnet*
| *Event-ib-alloc-Event*
| *Event-ib-free-Event*

datatype *Event-Path* = *Event-path-unlink path dentry*
| *Event-path-mkdir path dentry nat*
| *Event-path-rmdir path dentry*
| *Event-path-mknod path dentry nat nat*
| *Event-path-truncate path*
| *Event-path-symlink path dentry string*
| *Event-path-link dentry path dentry*
| *Event-path-rename path dentry path*
| *Event-path-chmod path nat*
| *Event-path-chown path kuid kgid*
| *Event-path-chroot path*

datatype *Event-Key* =
Event-key-permission process-id key-ref-t Cred nat
| *Event-key-getsecurity process-id key-serial-t string int*

datatype *Event-audit* = *Event-audit-data-to-entry process-id*
| *Event-audit-dupe-lsm-field process-id audit-field audit-field*
| *Event-audit-rule-known process-id audit-krule*
| *Event-audit-rule-match process-id int*
| *Event-audit-rule-free process-id audit-field*

datatype *Event-file* =
Event-do-ioctl process-id Files IOC-DIR nat
| *Event-syscall-ioctl process-id nat IOC-DIR nat*
| *Event-ksys-ioctl process-id nat IOC-DIR nat*
| *Event-vm-mmap-pgoff process-id Files nat nat nat nat nat*
| *Event-do-sys-vm86 process-id*
| *Event-get-unmapped-area process-id Files nat*
| *Event-validate-mmap-request process-id Files nat*

| *Event-generic-setlease process-id Files int*
| *Event-syscall-lock process-id nat nat*
| *Event-do-lock-file-wait process-id Files int file-lock*
| *Event-file-fcntl process-id Files nat nat*

| *Event-file-send-sigiotask process-id Task fown-struct int*
| *Event-file-receive process-id Files*

| *Event-do-dentry-open process-id Files*

datatype *Event-d-instantiate-procattr = Event-d-instantiate process-id dentry inode option*
| *Event-d-instantiate-new process-id dentry inode*
| *Event-d-instantiate-anon′ process-id dentry inode bool*
| *Event-d-add process-id dentry inode option*
| *Event-d-splice-alias process-id inode dentry*
| *Event-nfs-get-root process-id super-block nfs-fh string*
| *Event-proc-pid-attr-read process-id Files string nat loff-t*
| *Event-proc-pid-attr-write process-id Files string nat loff-t*

datatype *Event-ismaclabel = Event-nfs4-xattr-set-nfs4-label process-id xattr inode string*
| *Event-nfs4-xattr-get-nfs4-label process-id xattr inode string*

datatype *Event-id2secctx =*
Event-scm-passec process-id socket msghdr scm-cookie
| *Event-audit-receive-msg process-id sk-buff nlmsghdr*
| *Event-audit-log-name process-id audit-names*
| *Event-audit-log-task-context process-id audit-buffer*
| *Event-audit-log-pid-context process-id u32*
| *Event-show-special process-id audit-context*
| *Event-ctnetlink-dump-secctx process-id sk-buff nf-conn*
| *Event-ctnetlink-secctx-size process-id nf-conn*
| *Event-ct-show-secctx process-id seq-file nf-conn*
| *Event-nfqnl-get-sk-secctx process-id sk-buff string*
| *Event-netlbl-unlhsh-func3 process-id u32*
| *Event-netlbl-unlabel-staticlist-gen process-id u32*
| *Event-netlbl-audit-start-common process-id int netlbl-audit*

datatype *Event-secctxtoid =*
Event-set-security-override-from-ctx process-id Cred string
| *Event-netlbl-unlabel-staticadd process-id*

datatype *Event-inodesecctx =*
Event-kernfs-refresh-inode process-id kernfs-node inode
| *Event-nfs-setsecurity process-id inode nfs4-label*
| *Event-nfsd4-security-inode-setsecctx process-id svc-fh xdr-netobj u32*
| *Event-nfsd4-set-nfs4-label process-id svc-fh xdr-netobj*
| *Event-nfsd4-encode-fattr process-id dentry*
| *Event-ovl-get-tmpfile process-id ovl-copy-up-ctx*
| *Event-ovl-copy-xattr process-id dentry dentry*
| *Event-ovl-create-or-link process-id dentry inode ovl-cattr bool*

datatype *Event-others = Evt-ismaclabel Event-ismaclabel* | *Evt-id2secctx Event-id2secctx*

| *Evt-secctxtoid Event-secctxtoid* | *Evt-inodesecctx Event-inodesecctx*

definition *result s f* $\equiv$ *if fst* (*the-run-state f s*) $\neq$ *0 then False else True*

definition *resultU s f* $\equiv$ *if fst* (*the-run-state f s*) = () *then True else False*

definition *resultValue s f* $\equiv$ *fst* (*the-run-state f s*)

definition *funcState s f* $\equiv$ *snd* (*the-run-state f s*)

definition *getsb-by-id s i* $\equiv$ *the*((*k-superblock s*) *i*)

definition *getsb-id s sb* $\equiv$ *SOME i*. *sb* = *the*((*k-superblock s*) *i*)

29.2 kernel action about superblock

kernel create new superblock Allocate and attach a security structure if operation was successful created superblock else create fail

29.2.1 kernel action of security$_{s}b_{a}lloc$

definition *k-create-new-superblock* :: *'a* $\Rightarrow$ *process-id* $\Rightarrow$ *t-sb* $\Rightarrow$ (*'a* $\times$ *super-block option*)
  where *k-create-new-superblock s pid p* $\equiv$
    let t = *getsb-by-id s p*
    in
    if result s (*security-sb-alloc s t*) *then*
      (*snd* (*the-run-state* (*security-sb-alloc s t*) *s*), *Some t*)
    else
      (*s*, *None*)

kernel free superblock sb Deallocate and clear the sb security field.

29.2.2 kernel action of security$_{s}b_{f}ree$

definition *k-sb-free* :: *'a* $\Rightarrow$ *process-id* $\Rightarrow$ *t-sb* $\Rightarrow$ (*'a* $\times$ *unit*)
  where *k-sb-free s pid t* $\equiv$
    let s'= *snd*(*the-run-state*(*security-sb-free s* ((*getsb-by-id s t*))) *s*)
    in (*s'*,())

29.2.3 kernel action of security$_{s}b_{c}opy_{d}ata$

definition *k-sb-copy-data*:: *'a* $\Rightarrow$ *process-id* $\Rightarrow$*'a* $\times$ *int*
  where *k-sb-copy-data s pid* $\equiv$
    let copy = *SOME x*::*string*. *True*; *orig* = []
    in
    if result s (*security-sb-copy-data s orig copy*)
    then (*s*,*resultValue s* (*security-sb-copy-data s orig copy*))
    else (*s*,*0*)

29.2.4 kernel action of security$_{s}b_{r}emount$

Extracts security system specific mount options and verifies no changes are being made to those options.

definition *do-remount* :: $'a \Rightarrow path \Rightarrow Void \Rightarrow ('a \times int)$
where *do-remount s p data* $\equiv$
if result s (security-sb-remount s (mnt-sb (p-mnt p)) data)
then (s, resultValue s (security-sb-remount s (mnt-sb (p-mnt p)) data))
else (s,0)

29.2.5 kernel action of security$_{s}b_{k}ern_{m}ount$

definition *mount-fs*:: $'a \Rightarrow$ *file-system-type* $\Rightarrow int \Rightarrow string$
$\Rightarrow string \Rightarrow ('a \times$ *dentry option*)
where *mount-fs s type f name data* $\equiv$
let
secdata = (*SOME x*:: *string* . *True*);
root = (*SOME x*:: *dentry* . *True*) ;
t = *getsb-id s* (*d-sb root*)
in
if $\neg$(*result s* (*security-sb-copy-data s data secdata*)
$\wedge$ *result s* (*security-sb-kern-mount s* (*d-sb root*) *f secdata*))
then
(*s*, *Some root*)
else
(*s*,*None*)

29.2.6 kernel action of security$_{s}b_{s}how_{o}ptions$

definition *show-sb-opts* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *seq-file* $\Rightarrow$ *t-sb* $\Rightarrow 'a \times int$
where *show-sb-opts s pid m t* $\equiv$
(*s*, *resultValue s* (*security-sb-show-options s m* (*getsb-by-id s t*)))

29.2.7 kernel action of security$_{s}b_{s}tatfs$

definition *statfs-by-dentry* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *dentry* $\Rightarrow 'a \times int$
where *statfs-by-dentry s pid d* $\equiv$ (*s*,*resultValue s* (*security-sb-statfs s d*))

29.2.8 kernel action of security$_{s}b_{m}ount$

definition *do-mount* :: $'a \Rightarrow$ *process-id* $\Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow nat \Rightarrow Void$
$\Rightarrow 'a \times int$
where *do-mount s pid dev-name dir-name type-page flags' data-page* $\equiv$
let p = *SOME x*:: *path*. *True*
in
(*s*, *resultValue s* (*security-sb-mount s dev-name p type-page flags' data-page*))

29.2.9 kernel action of security$_{s}b_{u}mount$

definition *do-umount* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *mount* $\Rightarrow int \Rightarrow 'a \times int$

where *do-umount s pid m f* ≡
let $m' = (mnt\ m)$
in
$(s, resultValue\ s\ (\ security\text{-}sb\text{-}umount\ s\ m'\ f))$

29.2.10 kernel action of security$_s b_p ivotroot$

definition *pivot-root* :: *'a* ⇒ *process-id* ⇒ *'a* × *int*
where *pivot-root s pid* ≡
let new = *SOME x*:: *path. True*;
old = *SOME x*:: *path. True*
in
(*s*, *resultValue s* (*security-sb-pivotroot s new old*))

29.2.11 kernel action of security$_s b_s et_m nt_o pts$

definition *set-sb-security* :: *'a* ⇒ *process-id* ⇒ *super-block* ⇒ *dentry*
⇒ *nfs-mount-info* ⇒ *'a* × *int*
where *set-sb-security s pid sb d nfs* ≡
let opt = *lsm-opts* (*parsed nfs*);
kflags = *0* ;
kflags-out = *0*
in
(*s*,*resultValue s* (*security-sb-set-mnt-opts s sb opt kflags kflags-out*))

definition *setup-security-options* :: *'a* ⇒ *process-id* ⇒ *btrfs-fs-info* ⇒ *super-block*
⇒ *opts* ⇒ *'a* × *int*
where *setup-security-options s pid fsinfo sb sec-opts* ≡
(*s*,*resultValue s* (*security-sb-set-mnt-opts s sb sec-opts 0 0*))

29.2.12 kernel action of security$_s b_c lone_m nt_o pts$

definition *nfs-clone-sb-security* :: *'a* ⇒ *process-id* ⇒ *super-block* ⇒ *dentry* ⇒
nfs-mount-info => *'a* × *int*
where *nfs-clone-sb-security s pid sb' mntroot minfo* ≡
let oldsb = *nfsc-sb* (*cloned minfo*);
kflags = *0*;
kflags-out = *0*
in if result s (*security-sb-clone-mnt-opts s oldsb sb' kflags kflags-out*) *then*
(*s*,*resultValue s* (*security-sb-clone-mnt-opts s oldsb sb' kflags kflags-out*))
else
(*s*,*0*)

29.2.13 kernel action of security$_s b_p arse_o pts_s tr$

definition *parse-security-options* :: *'a* ⇒ *process-id* ⇒ *string* ⇒ *opts* ⇒ *'a* × *int*
where *parse-security-options s pid orig sec-opts* ≡
let secdata = *SOME x*:: *string. True*
in

$(s, resultValue\ s\ (security\text{-}sb\text{-}parse\text{-}opts\text{-}str\ s\ \ secdata\ sec\text{-}opts))$

29.3 task

29.3.1 kernel action of security$_{t}ask_{a}lloc$

definition *copy-process* :: $'a \Rightarrow process\text{-}id \Rightarrow nat \Rightarrow 'a \times Task\ option$
where *copy-process s pid cflags* $\equiv$
let $t = SOME\ x{::}\ Task.\ True;\ p = SOME\ x :: nat.\ True$ *in*
if result s $(security\text{-}task\text{-}alloc\ s\ t\ cflags)$
then $(s, None)$
else
$(funcState\ s\ (security\text{-}task\text{-}alloc\ s\ t\ cflags)\ , Some\ t)$

29.3.2 kernel action of security$_{t}ask_{f}ree$

definition *task-free* :: $'a \Rightarrow process\text{-}id \Rightarrow Task \Rightarrow 'a \times unit$
where *task-free s p t* $\equiv$
let $pid = get\text{-}processid\ s\ t$;
$s' = snd(the\text{-}run\text{-}state(security\text{-}task\text{-}free\ s\ t)\ s)$
in $(s', ())$

29.3.3 kernel action of security$_{c}red_{a}lloc_{b}lank$

definition *cred-alloc-blank* :: $'a \Rightarrow process\text{-}id \Rightarrow 'a \times Cred\ option$
where *cred-alloc-blank s pid* $\equiv$ *let* $new = SOME\ x{::}\ Cred.\ True$ *in*
if $resultValue\ s\ (security\text{-}cred\text{-}alloc\text{-}blank\ s\ new\ 0) \leq 0$ *then*
$(s, None)$
else
$(s, Some\ new)$

29.3.4 kernel action of security$_{c}red_{f}ree$

definition *cred-free* :: $'a \Rightarrow process\text{-}id \Rightarrow 'a \times unit$
where *cred-free s pid* $\equiv$
let $cred\ = SOME\ x{::}\ Cred.\ True$;
$s' = snd(the\text{-}run\text{-}state(security\text{-}cred\text{-}free\ s\ cred)\ s)$
in $(s', ())$

29.3.5 kernel action of security$_{p}repare_{c}reds$

definition *prepare-creds* :: $'a \Rightarrow process\text{-}id \Rightarrow 'a \times Cred\ option$
where *prepare-creds s pid* $\equiv$
let $task = current\text{-}process\ s$;
$new = SOME\ x{::}\ Cred.\ True$;
$old = cred\ task$
in
if $resultValue\ s\ (security\text{-}prepare\text{-}creds\ s\ new\ old\ 0) < 0$ *then* $(s, None)$
else $(s, Some\ new)$

29.3.6 kernel action of security$_{t}ransfer_{c}reds$

definition *key-change-session-keyring* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ $'a\times$ *unit*
where *key-change-session-keyring s pid* $\equiv$
let new = *SOME x*:: *Cred*. *True*;
old = *current-cred s*;
s' = *snd*(*the-run-state*(*security-transfer-creds s new old*) *s*)
in (*s'*,())

29.3.7 kernel action of security$_{t}ask_{f}ix_{s}etuid$

definition *sys-setreuid* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *kuid* $\Rightarrow$ *kuid* $\Rightarrow$ $'a \times$ *int*
where *sys-setreuid s pid ruid euid'* $\equiv$
let new = *snd*(*prepare-creds s pid*);
old = *current-cred s* ;
retval = *resultValue s* (*security-task-fix-setuid s* (*the new*)
old LSM-SETID-RE)
in
if new = *None then*
(*s*,$-$*ENOMEM*)
else
if retval < *0 then*
(*s*,*retval*)
else
(*s*,*0*)

29.3.8 kernel action of security$_{t}ask_{s}etpgid$

definition *setpgid* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *pid-t* $\Rightarrow$ *pid-t* $\Rightarrow$ $'a \times$ *int*
where *setpgid s p pid pgid* $\equiv$
let pgid = *if pgid* = *0 then pid else pgid*;
p = *get-process-by-pid s* (*nat pid*) *in*
if pgid < *0 then*
(*s*,$-$*EINVAL*)
else
let err = *resultValue s* (*security-task-setpgid s p pgid*)
in
if err $\neq$ *0 then*
(*s*,*err*)
else
(*s*,*0*)

29.3.9 kernel action of security$_{t}ask_{g}etpgid$

definition *do-getpgid* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*pid-t* $\Rightarrow$ $'a \times$ *int*
where *do-getpgid s p pid* $\equiv$
let p = *get-process-by-pid s* (*nat pid*);
retval = *resultValue s* (*security-task-getpgid s p*)
in if retval $\neq$ *0 then*
(*s*, *retval*)

else
  $(s,\ pid)$

29.3.10 kernel action of security$_{t}ask_{g}etsid$

definition *getsid* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *pid-t* $\Rightarrow$ $'a \times$ *int*
  where *getsid s p pid* $\equiv$
    if pid $=$ *0 then* $(s,$*current s*$)$
    else
      let p $=$ *get-process-by-pid s* $($*nat pid*$)$;
      retval $=$ *resultValue s* $($ *security-task-getsid s p*$)$
      in
      if retval $\neq$ *0 then*
        $(s,$*retval*$)$
      else
        $(s,$*pid*$)$

29.3.11 kernel action of security$_{t}ask_{g}etsecid$

definition *getsecid* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Task* $\Rightarrow$ *u32* $\Rightarrow$ $'a \times$ *unit*
  where *getsecid s pid p secid'* $\equiv$
    let secid' $=$ *0*;
      retval $=$ *resultValue s* $($ *security-task-getsecid s p secid'*$)$
    in $(s,$*retval*$)$

29.3.12 kernel action of security$_{t}ask_{s}etnice$

definition *task-setnice* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Task* $\Rightarrow$*int* $\Rightarrow$ $'a \times$ *int*
  where *task-setnice s pid p nice* $\equiv$
    let
      retval $=$ *resultValue s* $($ *security-task-setnice s p nice*$)$
    in if retval $\neq$ *0 then*
        $(s,$*retval*$)$
      else
        $(s,$*0*$)$

29.3.13 kernel action of security$_{t}ask_{s}etioprio$

definition *set-task-ioprio* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Task* $\Rightarrow$ *int* $\Rightarrow$ $'a \times$ *int*
  where *set-task-ioprio s pid p ioprio* $\equiv$
    let
      retval $=$ *resultValue s* $($ *security-task-setioprio s p ioprio*$)$
    in if retval $\neq$ *0 then* $(s,$*retval*$)$ *else* $(s,$*0*$)$

29.3.14 kernel action of security$_{t}ask_{g}etioprio$

definition *get-task-ioprio* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Task* $\Rightarrow$ $'a \times$ *int*
  where *get-task-ioprio s pid p* $\equiv$
    let
      retval $=$ *resultValue s* $($ *security-task-getioprio s p* $)$

in if retval ≠ 0 then (s,retval) else (s,0)

29.3.15 kernel action of security$_{t}ask_{p}rlimit$

definition *check-prlimit-permission* :: *'a ⇒ process-id ⇒ Task ⇒ nat ⇒ 'a × int*
  where *check-prlimit-permission s pid p flags' ≡*
    let current = current-process s;
      cred = current-cred s;
      tcred = task-cred s p
    in
     if current = p then
       (s,0)
     else
       (s,resultValue s (security-task-prlimit s cred tcred flags'))

29.3.16 kernel action of security$_{t}ask_{s}etrlimit$

definition *do-prlimit* :: *'a ⇒ process-id ⇒ Task ⇒ nat ⇒ 'a × int*
  where *do-prlimit s pid p resource ≡*
    let new-rlim = SOME x:: rlimit. True
    in (s,resultValue s (security-task-setrlimit s p resource new-rlim))

29.3.17 kernel action of security$_{t}ask_{s}etscheduler$

definition *task-setscheduler* :: *'a ⇒ process-id ⇒ Task ⇒ 'a × int*
  where *task-setscheduler s pid p ≡*
    let retval = resultValue s (security-task-setscheduler s p)
    in if retval ≠ 0 then
        (s,retval)
       else (s,0)

29.3.18 kernel action of security$_{t}ask_{g}etscheduler$

definition *task-getscheduler* :: *'a ⇒ process-id ⇒ Task ⇒ 'a × int*
  where *task-getscheduler s pid p ≡*
    let retval = resultValue s (security-task-getscheduler s p)
    in if retval ≠ 0 then (s,retval) else (s,0)

29.3.19 kernel action of security$_{t}ask_{m}ovememory$

definition *task-movememory* :: *'a ⇒ process-id ⇒ Task ⇒ 'a × int*
  where *task-movememory s pid p ≡*
    let retval = resultValue s (security-task-movememory s p)
    in if retval ≠ 0 then (s,retval) else (s,0)

29.3.20 kernel action of security$_{t}ask_{k}ill$

definition *task-kill* :: *'a ⇒ process-id ⇒ Task ⇒ siginfo ⇒int ⇒ Cred⇒ 'a × int*
  where *task-kill s pid p info sig c ≡*

let retval = resultValue s (security-task-kill s p info sig (Some c))
in if retval ≠ 0 then (s,retval)
else (s,0)

29.3.21 kernel action of security$_task_prctl$

definition *task-prctl* :: *'a ⇒ process-id⇒ int ⇒ nat ⇒nat ⇒nat ⇒nat⇒ 'a × int*
where *task-prctl s pid op arg2 arg3 arg4 arg5 ≡*
let retval = resultValue s (security-task-prctl s op arg2 arg3 arg4 arg5)
in if retval ≠ (−ENOSYS) then
(s,retval)
else
(s,0)

29.3.22 kernel action of security$_task_getsecid$

definition *ima-bprm-check'* :: *'a ⇒ process-id ⇒ linux-binprm ⇒ 'a × int*
where *ima-bprm-check' s pid bprm ≡*
let secid = SOME x:: u32. True;
ret = security-task-getsecid s (current-process s) secid
in (s,0)

29.3.23 kernel action of security$_kernel_act_as$

definition *set-security-override* :: *'a ⇒ process-id⇒Cred ⇒ u32 ⇒ 'a × int*
where *set-security-override s pid new secid' ≡*
let retval = resultValue s (security-kernel-act-as s new secid')
in (s,retval)

29.3.24 kernel action of security$_kernel_create_files_as$

definition *set-create-files-as* :: *'a ⇒ process-id⇒Cred ⇒ inode ⇒ 'a × int*
where *set-create-files-as s pid new inode ≡*
let
new = new (|fsuid := i-uid inode,fsgid := i-gid inode|);
retval = resultValue s (security-kernel-create-files-as s new inode)
in (s,retval)

29.3.25 kernel action of security$_kernel_module_request$

definition *request-module'* :: *'a ⇒ process-id ⇒ 'a × int*
where *request-module' s pid ≡*
let
module-name = SOME x::string. True;
retval = resultValue s (security-kernel-module-request s module-name)
in (s,retval)

29.3.26 kernel action of security$_kernel_read_file$

definition *kernel-read-file* :: *'a ⇒ process-id ⇒ Files ⇒ string*

$\Rightarrow$ *kernel-read-file-id* $\Rightarrow$ *'a* $\times$ *int*
where *kernel-read-file s pid file buf id'* $\equiv$
let
retval = *resultValue s* (*security-kernel-read-file s file id'*)
in if retval $\neq$ *0 then* (*s*,*retval*)
else
let
i-size' = *nat*(*ii-size* (*file-inode file*));
retval = *resultValue s* (*security-kernel-post-read-file s file buf i-size' id'*)
in (*s*,*retval*)

29.3.27 kernel action of security$_kernel_load_data$

definition *load-data* :: *'a* $\Rightarrow$ *process-id*$\Rightarrow$ *'a* $\times$ *int*
where *load-data s pid* $\equiv$
let
load = *SOME x*::*kernel-load-data-id*. *True*;
retval = *resultValue s* (*security-kernel-load-data s load*)
in if retval $\neq$ *0 then* (*s*,*retval*) *else* (*s*,*0*)

29.3.28 kernel action of security$_task_to_inode$

definition *task-to-inode* :: *'a* $\Rightarrow$ *process-id* $\Rightarrow$ *Task* $\Rightarrow$ *inode* $\Rightarrow$ *'a* $\times$ *unit*
where *task-to-inode s pid task inode* $\equiv$
let
load = *SOME x*::*kernel-load-data-id*. *True*;
s' = *funcState s* (*security-task-to-inode s task inode*)
in (*s*,())

29.3.29 kernel action of security$_getprocattr$

definition *PROC-I* :: *inode* $\Rightarrow$ *proc-inode*
where *PROC-I inode* $\equiv$ *SOME proc* . *vfs-inode proc* = *inode*

definition *proc-pid* :: *inode* $\Rightarrow$ *ppid*
where *proc-pid inode* $\equiv$ *proci-pid* (*PROC-I inode*)

definition *get-pid-task* :: *'a* $\Rightarrow$*ppid* $\Rightarrow$ *Task*
where *get-pid-task s p* $\equiv$ *the* ((*k-task s*)(*tid p*))

definition *proc-pid-attr-read* :: *'a* $\Rightarrow$ *process-id* $\Rightarrow$ *Files* $\Rightarrow$ *string* $\Rightarrow$ *nat* $\Rightarrow$ *loff-t* $\Rightarrow$ *'a* $\times$ *int*
where *proc-pid-attr-read s pid file buf count' ppos* $\equiv$
let
p = *SOME x*:: *string*. *True*;
inode = *file-inode file*;
ppid' = *proc-pid inode*;
task = *get-pid-task s ppid'*;

retval = resultValue s (security-getprocattr s task (d-name(p-dentry(f-path file))) p)
in (s,retval)

29.3.30 kernel action of security$_{s}etprocattr$

definition *proc-pid-attr-write*:: $'a \Rightarrow$ *process-id*$\Rightarrow$ *Files*$\Rightarrow$ *string* $\Rightarrow$ *nat* $\Rightarrow$ *loff-t*$\Rightarrow'a \times$ *int*
where*proc-pid-attr-write s pid file buf count' ppos* $\equiv$
let
p = SOME x:: string. True;
inode = file-inode file;
ppid' = proc-pid inode;
task = get-pid-task s ppid';
name = (d-name(p-dentry(f-path file)));
retval = resultValue s (security-setprocattr s name p (int count'))
in (s,retval)

29.4 binder

29.4.1 kernel action of security$_{b}inder_{s}et_{c}ontext_{m}gr$

definition *binder-ioctl-set-ctx-mgr* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Files* $\Rightarrow$ $'a \times$ *int*
where *binder-ioctl-set-ctx-mgr s pid files'* $\equiv$
let proc = private-data files';
task = tsk proc ;
retval = resultValue s (security-binder-set-context-mgr s task)
in if retval < 0 then
(s,retval)
else
(s,0)

29.4.2 kernel action of security$_{b}inder_{t}ransaction$

definition *binder-transaction* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *binder-proc*
$\Rightarrow$ *binder-thread* $\Rightarrow$ $'a \times$ *unit*
where *binder-transaction s pid proc' thread* $\equiv$
let
task = tsk proc';
target-task = tsk (proc thread);
retval = resultValue s (security-binder-transaction s task target-task)
in if retval < 0 then
(s,())
else
(s,())

29.4.3 kernel action of security$_{b}inder_{t}ransfer_{b}inder$

definition *binder-translate-binder* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *flat-binder-object*

$\Rightarrow$ *binder-transaction* $\Rightarrow$ *binder-thread* $\Rightarrow$ *'a* $\times$ *int*
where *binder-translate-binder s pid fp t thread* $\equiv$
let
target-task = *tsk* (*to-proc t*) ;
task = *tsk* (*proc thread*);
retval = *resultValue s* (*security-binder-transfer-binder s task target-task*)
in if retval $\neq$ *0 then*
(*s*,−*EPERM*)
else
(*s*,*0*)

29.4.4 kernel action of security$_binder_transfer_file$

definition *binder-translate-fd* :: *'a* $\Rightarrow$ *process-id* $\Rightarrow$ *int* $\Rightarrow$ *binder-transaction* $\Rightarrow$ *binder-thread*
$\Rightarrow$ *binder-transaction* $\Rightarrow$ *'a* $\times$ *int*
where *binder-translate-fd s pid fd t thread in-reply-to* $\equiv$
let
target-task = *tsk* (*to-proc t*) ;
task = *tsk* (*proc thread*);
f = *SOME x* :: *Files. True*;
retval = *resultValue s* (*security-binder-transfer-file s task target-task f*)
in if retval < *0 then*
(*s*,−*EPERM*)
else
(*s*,*0*)

29.5 ptrace sys

29.5.1 kernel action of security$_ptrace_access_check$

definition *ptrace-may-access* :: *'a* $\Rightarrow$ *process-id* $\Rightarrow$ *Task* $\Rightarrow$ *nat* $\Rightarrow$ *'a* $\times$ *int*
where *ptrace-may-access s pid task m* $\equiv$
let retval = *resultValue s* (*security-ptrace-access-check s task m*)
in (*s*,*retval*)

29.5.2 kernel action of security$_ptrace_traceme$

definition *ptrace-traceme* :: *'a* $\Rightarrow$ *process-id* $\Rightarrow$ *'a* $\times$ *int*
where *ptrace-traceme s pid* $\equiv$
if ptrace (*current-process s*) = *0*
then (*s*,−*EPERM*)
else
let parent = *get-process-by-pid s* (*parent*(*current-process s*));
retval = *resultValue s* (*security-ptrace-traceme s parent*)
in (*s*,*retval*)

29.5.3 kernel action of security$_syslog$

definition *check-syslog-permissions* :: *'a* $\Rightarrow$ *process-id* $\Rightarrow$ *int* $\Rightarrow$ *'a* $\times$ *int*

where *check-syslog-permissions s pid t* ≡
let retval = *resultValue s* (*security-syslog s t*)
in (*s*,*retval*)

29.5.4 kernel action of security$_q uotactl$

definition *check-quotactl-permission* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*super-block* $\Rightarrow$*int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$$'a \times$ *int*
where *check-quotactl-permission s pid sb type' cmd id'*≡
let retval = *resultValue s* (*security-quotactl s cmd type' id'*(*Some sb*))
in (*s*,*retval*)

definition *quota-sync-all* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*int* $\Rightarrow$$'a \times$ *int*
where *quota-sync-all s pid t* ≡
let retval = *resultValue s* (*security-quotactl s Q-SYNC t 0 None*)
in (*s*,*retval*)

29.5.5 kernel action of security$_q uota_o n$

definition *dquot-quota-on* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*super-block* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$*path* $\Rightarrow$$'a \times$ *int*
where *dquot-quota-on s pid sb type' fromat-id path* ≡
let retval = *resultValue s* (*security-quota-on s* (*p-dentry path*))
in (*s*,*retval*)

definition *dquot-quota-on-mount* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*super-block* $\Rightarrow$ *string* $\Rightarrow$ *int* $\Rightarrow$*int* $\Rightarrow$$'a \times$ *int*
where *dquot-quota-on-mount s pid sb qf-name fromat-id type'* ≡
let dentry = *SOME x* :: *dentry*. *True*;
retval = *resultValue s* (*security-quota-on s dentry*)
in (*s*,*retval*)

29.5.6 kernel action of security$_s ettime64$

definition *syscall-stime* :: $'a \Rightarrow$ *process-id*$\Rightarrow$$'a \times$ *int*
where *syscall-stime s pid* ≡
let tv = *SOME x* :: *timespec64*. *True*;
retval = *resultValue s* (*security-settime64 s tv None*)
in if retval ≠ *0 then* (*s*,*retval*) *else* (*s*,*0*)

definition *do-sys-settimeofday64* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *timespec64* $\Rightarrow$ *tz* $\Rightarrow$$'a \times$ *int*
where *do-sys-settimeofday64 s pid tv tz* ≡
let
retval = *resultValue s* (*security-settime64 s tv* (*Some tz*))
in if retval ≠ *0 then* (*s*,*retval*) *else* (*s*,*0*)

type-synonym *pages* = *int*

29.5.7 kernel action of security$_v m_e nough_m emory_m m$

definition *frontswap-unuse-pages* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *int* $\Rightarrow 'a \times$ *int*
where *frontswap-unuse-pages s pid total' unused swapid* $\equiv$
let pages = SOME x :: pages. True;
mm = mm (current-process s);
retval = resultValue s (security-vm-enough-memory-mm s mm pages)
in if retval $\neq$ *0 then (s,−ENOMEM) else (s,0)*

definition *vma-pages* :: *vm-area-struct* $\Rightarrow$ *nat*
where *vma-pages vma* $\equiv$ *nat((int (vm-end vma − vm-start vma)) >> PAGE-SHIFT)*

definition *latent-entropy* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *mm* $\Rightarrow$ *mm* $\Rightarrow 'a \times$ *int*
where *latent-entropy s pid mm' oldmm* $\equiv$
let pages = SOME x :: pages. True;
len = vma-pages(mmap oldmm);
retval = resultValue s (security-vm-enough-memory-mm s oldmm pages)

in if retval $\neq$ *0 then (s,−ENOMEM) else (s,0)*

definition *mmap-region* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *nat* $\Rightarrow 'a \times$ *int*
where *mmap-region s pid len'* $\equiv$
let mm = mm (current-process s);
charged =(len' >>PAGE-SHIFT);
retval = resultValue s (security-vm-enough-memory-mm s mm charged)
in if retval $\neq$ *0 then (s,−ENOMEM) else (s,0)*

definition *acct-stack-growth* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *vm-area-struct* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow 'a \times$ *int*
where *acct-stack-growth s pid vma size' grow* $\equiv$
let mm = SOME x :: mm. True;
retval = resultValue s (security-vm-enough-memory-mm s mm grow)
in if retval $\neq$ *0 then (s,−ENOMEM) else (s,0)*

definition *do-brk-flags* :: $'a \Rightarrow$ *process-id*$\Rightarrow$ *nat* $\Rightarrow 'a \times$ *int*
where *do-brk-flags s pid len'* $\equiv$
let mm = mm (current-process s);
charged =(len' >> PAGE-SHIFT);
retval = resultValue s (security-vm-enough-memory-mm s mm charged)
in if retval $\neq$ *0 then (s,−ENOMEM) else (s,0)*

definition *insert-vm-struct* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *mm* $\Rightarrow$ *vm-area-struct* $\Rightarrow 'a \times$ *int*
where *insert-vm-struct s pid mm' vma* $\equiv$
let pages = SOME x :: pages. True;
len = vma-pages(vma);
retval = resultValue s (security-vm-enough-memory-mm s mm'

pages)
  in if retval $\neq$ *0 then* (*s*,−*ENOMEM*) *else* (*s*,*0*)

definition *mprotect-fixup* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *vm-area-struct* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ $'a \times$ *int*
  where *mprotect-fixup s pid vma end start* $\equiv$
    let mm = *SOME x* :: *mm*. *True*;
      len = *end* − *start*;
      nrpages =(*len* >> *PAGE-SHIFT*);
      retval = *resultValue s* (*security-vm-enough-memory-mm s mm nrpages*)
    in if retval $\neq$ *0 then* (*s*,−*ENOMEM*) *else* (*s*,*0*)

definition *vma-to-resize* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ $'a \times$ *vm-area-struct option*
  where *vma-to-resize s pid addr old-len new-len p* $\equiv$
    let mm = *mm* (*current-process s*);
      len = *old-len* − *new-len*;
      charged =(*len* >> *PAGE-SHIFT*);
      vma = *SOME x* :: *vm-area-struct*. *True*;
      retval = *resultValue s* (*security-vm-enough-memory-mm s mm charged*)
    in if retval $\neq$ *0 then* (*s*,*None*) *else* (*s*,*Some vma*)

definition *PAGE-MASK* $\equiv$ *NOT* (*PAGE-SIZE* − *1*)
definition *PAGE-ALIGN addr* $\equiv$ (*addr* + *PAGE-SIZE* − *1*) *AND PAGE-MASK*
definition *VM-ACCT size'* $\equiv$ *PAGE-ALIGN*(*size'*) >> *PAGE-SHIFT*

definition *shmem-acct-size* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *nat* $\Rightarrow$ *loff-t* $\Rightarrow 'a \times$ *int*
  where *shmem-acct-size s pid flags' size'* $\equiv$
    let mm = *mm* (*current-process s*);
      charged = *VM-ACCT size'*;
      retval = (*if* ((*int flags'*) *AND VM-NORESERVE*) $\neq$ *0 then 0*
        else
        resultValue s (*security-vm-enough-memory-mm s mm charged*))
    in (*s*,*retval*)

definition *shmem-reacct-size* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *nat* $\Rightarrow$ *loff-t* $\Rightarrow$ *loff-t* $\Rightarrow$ $'a \times$ *int*
  where *shmem-reacct-size s pid flags' oldsize newsize* $\equiv$
    let mm = *mm* (*current-process s*);
      charged = *VM-ACCT newsize* − *VM-ACCT oldsize*;
      retval = (*if charged* > *0 then*
        resultValue s (*security-vm-enough-memory-mm s mm charged*)
        else
          0)
    in (*s*,*retval*)

definition *shmem-acct-block* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow 'a \times$ *int*
where *shmem-acct-block s pid flags' pages'* $\equiv$
let mm = *mm* (*current-process s*);
charged = *pages'* $*$ (*VM-ACCT PAGE-SIZE*) ;
retval = (*if* ((*int flags'*) *AND VM-NORESERVE*) $\neq$ *0 then 0*
else
resultValue s (*security-vm-enough-memory-mm s mm charged*))
in (*s*,*retval*)

definition *syscall-swapoff* :: $'a \Rightarrow$ *process-id* $\Rightarrow 'a \times$ *int*
where *syscall-swapoff s pid* $\equiv$
let mm = *mm* (*current-process s*);
pages = *SOME x* :: *pages. True*;
retval = *resultValue s* (*security-vm-enough-memory-mm s mm pages*)
in if retval $\neq$ *0 then* (*s*,−*ENOMEM*) *else* (*s*,*0*)

29.6 cap

29.6.1 kernel action of security$_c$*apget*

definition *cap-get-target-pid* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *kernel-cap-t* $\Rightarrow$ *kernel-cap-t* $\Rightarrow$ *kernel-cap-t*
$\Rightarrow 'a \times$ *int*
where *cap-get-target-pid s pid pEp pIp pPp* $\equiv$
let task = *SOME x* :: *Task. True*;
retval = *resultValue s* (*security-capget s task pEp pIp pPp*)
in (*s*,*retval*)

29.6.2 kernel action of security$_c$*apset*

definition *kcapset* :: $'a \Rightarrow$ *process-id* $\Rightarrow 'a \times$ *int*
where *kcapset s pid* $\equiv$
let task = *SOME x* :: *Task. True*;
effective = *SOME x* :: *kernel-cap-t. True*;
inheritable = *SOME x* :: *kernel-cap-t. True*;
permitted = *SOME x* :: *kernel-cap-t. True*;
new = (*the*(*snd*(*prepare-creds s pid*)));
old = *current-cred s*;
retval = *resultValue s* (*security-capset s new old effective inheritable permitted*)
in if retval < *0 then* (*s*,*retval*)
else (*s*,*0*)

29.6.3 kernel action of security$_c$*apable*

definition *has-ns-capability* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Task* $\Rightarrow$ *ns* $\Rightarrow$ *int* $\Rightarrow 'a \times$ *bool*
where *has-ns-capability s pid t ns cap* $\equiv$
let c = *task-cred s t*;
retval = *resultValue s* (*security-capable s c ns cap*)

in if retval = 0 then (s,True)
else (s,False)

definition *ns-capable-common* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *ns* $\Rightarrow$ *int* $\Rightarrow$ *bool* $\Rightarrow$ $'a \times$ *bool*
where *ns-capable-common s pid ns cap audit* $\equiv$
let c = current-cred s;
capable =
(if audit then resultValue s (security-capable s c ns cap)
else
resultValue s (security-capable-noaudit s c ns cap))
in if capable = 0 then
(s,True)
else
(s,False)

definition *file-ns-capable* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Files* $\Rightarrow$ *ns* $\Rightarrow$ *int* $\Rightarrow$ $'a \times$ *bool*
where *file-ns-capable s pid files' ns cap* $\equiv$
let c = f-cred files';
retval= resultValue s (security-capable s c ns cap)
in if retval $\neq$ *0 then*
(s,True)
else
(s,False)

29.6.4 kernel action of security$_capable_noaudit$

definition *has-ns-capability-noaudit* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Task* $\Rightarrow$ *ns* $\Rightarrow$ *int* $\Rightarrow$ $'a \times$ *bool*
where *has-ns-capability-noaudit s pid t ns cap* $\equiv$
let c = task-cred s t;
retval= resultValue s (security-capable-noaudit s c ns cap)
in if retval = 0 then
(s,True)
else
(s,False)

definition *ptracer-capable* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Task* $\Rightarrow$ *ns* $\Rightarrow$ $'a \times$ *bool*
where *ptracer-capable s pid t ns* $\equiv$
let c = ptracer-cred t;
retval= (if c = None then 0
else
resultValue s (security-capable-noaudit s (the c) ns CAP-SYS-PTRACE))
in if retval = 0 then
(s,True)
else
(s,False)

29.7 bprm

29.7.1 kernel action of security$_bprm_set_creds$

definition *prepare-binprm* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *linux-binprm* $\Rightarrow$ $'a \times$ *int*
where *prepare-binprm s pid bprm* $\equiv$
let
retval = *resultValue s* (*security-bprm-set-creds s bprm*)
in
if retval $\neq$ *0 then* (*s*,*retval*)
else (*s*,*0*)

29.7.2 kernel action of security$_bprm_check$

definition *search-binary-handler* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *linux-binprm* $\Rightarrow$ $'a \times$ *int*
where *search-binary-handler s pid bprm* $\equiv$
let
retval = *resultValue s* (*security-bprm-check s bprm*)
in
if retval $\neq$ *0 then* (*s*,*retval*)
else (*s*,$-$*ENOENT*)

29.7.3 kernel action of security$_bprm_committing_credssecurity_bprm_committed_creds$

definition *install-exec-creds* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *linux-binprm* $\Rightarrow$ $'a \times$ *unit*
where *install-exec-creds s pid bprm* $\equiv$
let
s' = *snd* (*the-run-state* (*security-bprm-committing-creds s bprm*) *s*);
s'' = *snd* (*the-run-state* (*security-bprm-committed-creds* s' *bprm*) s')
in
(s'',())

29.8 inode part 1

29.8.1 kernel action of security$_inode_alloc$

definition *inode-init-always* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *super-block* $\Rightarrow$ *inode* $\Rightarrow$ $'a \times$ *int*
where *inode-init-always s pid sb inode* $\equiv$
let inode = *inode*(|*i-opflags* := *0*,
i-sb := *sb*,
i-flags :=*0*|);
s' = *snd* (*the-run-state* (*security-inode-alloc s inode*) *s*);
retval = *resultValue s* (*security-inode-alloc s inode*)
in if retval $\neq$ *0 then*
(*s*,$-$*ENOMEM*)
else
(s',*0*)

29.8.2 kernel action of security$_inode_free$

definition *destroy-inode'* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *inode* $\Rightarrow$ $'a \times$ *unit*

where *destroy-inode′ s pid inode* ≡
let
s′ = *snd* (*the-run-state* (*security-inode-free s inode*) *s*)
in (*s′*,())

29.8.3 kernel action of security$_{d}entry_{i}nit_{s}ecurity$

definition *nfs4-label-init-security*:: *′a* ⇒ *process-id* ⇒ *inode* ⇒ *dentry* ⇒ *iattr* ⇒ *nfs4-label*
⇒ *′a* × *nfs4-label option*
where *nfs4-label-init-security s pid dir dentry sattr label′* ≡
let imode = *ia-mode sattr*;
dname = *d-name dentry*;
label = *label label′*;
len = *len label′*;
s′ = *snd* (*the-run-state* (*security-dentry-init-security s dentry imode dname label len*) *s*);
retval = *resultValue s* (*security-dentry-init-security s dentry imode dname label len*)
in if retval = *0 then*
(*s′*,*Some label′*)
else
(*s′*,*None*)

29.8.4 kernel action of security$_{d}entry_{c}reate_{f}iles_{a}s$

definition *override-creds* :: *′a* ⇒ *Cred* ⇒ *Cred option*
where *override-creds s new* ≡
let old = *current-cred s in Some old*

definition *ovl-override-creds* :: *′a* ⇒*super-block* ⇒ *Cred*
where *ovl-override-creds s sb* ≡
let ofs = *s-fs-info sb*
in the(*override-creds s* (*creator-cred ofs*))

definition *ovl-create-or-link*:: *′a* ⇒ *process-id* ⇒ *dentry* ⇒ *inode*
⇒ *ovl-cattr* ⇒ *bool* ⇒ *′a* × *int*
where *ovl-create-or-link s pid dentry inode attr′ origin* ≡
let
dname = *d-name dentry*;
mode = *mode attr′*;
old-cred = *ovl-override-creds s* (*d-sb dentry*);
override-cred = (*the*(*snd*(*prepare-creds s pid*))) ;
s′ = *snd* (*the-run-state* (*security-dentry-create-files-as s dentry mode dname old-cred override-cred*) *s*);
retval = *resultValue s* (*security-dentry-create-files-as s dentry mode dname old-cred override-cred*)
in if retval = *0 then*
(*s′*,*0*)
else

$(s',0)$

29.8.5 kernel action of security$_old_inode_init_security$

definition *ocfs2-init-security-get*:: $'a \Rightarrow$ *process-id* $\Rightarrow$ *inode* $\Rightarrow$ *inode* $\Rightarrow$ *string* $\Rightarrow$ *ocfs2-security-xattr-info option* $\Rightarrow$ $'a \times$ *int*
where *ocfs2-init-security-get s pid inode dir qstr si* $\equiv$
if si $\neq$ *None then*
let
name = *oname* (*the si*);
value = *vvalue* (*the si*);
len = *value-len* (*the si*);
s' = *snd* (*the-run-state* (*security-old-inode-init-security s inode dir qstr name value len*) *s*);
retval = *resultValue s* (*security-old-inode-init-security s inode dir qstr name value len*)
in (s',*retval*)
else
let
s' = *snd* (*the-run-state* (*security-inode-init-security s inode dir qstr 0* []) *s*);
retval = *resultValue s* (*security-inode-init-security s inode dir qstr 0* [])
in (s',*retval*)

definition *reiserfs-security-init*:: $'a \Rightarrow$ *process-id* $\Rightarrow$ *inode* $\Rightarrow$ *inode* $\Rightarrow$ *qstr* $\Rightarrow$ *reiserfs-security-handle* $\Rightarrow$ $'a \times$ *int*
where *reiserfs-security-init s pid inode dir qstr sec* $\equiv$
let
name = *rsh-name sec*;
value = *rsh-value sec*;
len = *rsh-len sec*;
s' = *snd* (*the-run-state* (*security-old-inode-init-security s inode dir qstr name value len*) *s*);
retval = *resultValue s* (*security-old-inode-init-security s inode dir qstr name value len*)
in (s',*retval*)

29.8.6 kernel action of security$_inode_init_security$

definition *xattr-security-init*:: $'a \Rightarrow$ *process-id* $\Rightarrow$ *inode* $\Rightarrow$ *inode* $\Rightarrow$ *qstr* $\Rightarrow$ *int* $\Rightarrow$ $'a \times$ *int*
where *xattr-security-init s pid inode dir qstr btrfs-initxattrs* $\equiv$
let
s' = *funcState s* (*security-inode-init-security s inode dir qstr btrfs-initxattrs* '''') ;
retval = *resultValue s* (*security-inode-init-security s inode dir qstr btrfs-initxattrs* '''')
in (s',*retval*)

29.9 path

29.9.1 kernel action of security$_{p}ath_{m}kdir$

definition *FSCACHE-COOKIE-TYPE-INDEX* $\equiv$ *0*

definition *container-of-cache* :: *fscache-cache* => *cachefiles-cache*
where *container-of-cache ptr* $\equiv$ *SOME type.* (*cache type*) = *ptr*

definition *cachefiles-walk-to-object* :: *'a* ⇒ *cachefiles-object* ⇒ *cachefiles-object* ⇒ *string*
⇒ *cachefiles-xattr* ⇒ *'a* × *int*
where *cachefiles-walk-to-object s parent' object key' auxdata* ≡
let cache = *container-of-cache* (*fsobj-cache* (*fscache parent'*));
path = *SOME x*:: *path* .*True* ;
dir = *co-dentry parent'*;
path = *path* (| *p-mnt* := *cc-mnt cache*, *p-dentry* := *dir*|);
next = *SOME x*:: *dentry* . *True*
in

if (*length*(*key'*) ≠ *0* ∨ (*co-type object* = *FSCACHE-COOKIE-TYPE-INDEX*))
then
let s' = *funcState s* (*security-path-mkdir s path next 0*);
retval = *resultValue s* (*security-path-mkdir s path next 0*)
in if retval < *0 then*
(*s'*,*retval*)
else
(*s'*,*0*)
else
let s' = *funcState s* (*security-path-mknod s path next S-IFREG 0*);
retval = *resultValue s* (*security-path-mknod s path next S-IFREG 0*)
in if retval < *0 then*
(*s'*,*retval*)
else
(*s'*,*0*)

29.9.2 kernel action of security$_{p}ath_{m}knodsecurity_{i}node_{c}reate$

definition *may-o-create* :: *'a* ⇒ *process-id* ⇒ *path* ⇒ *dentry* ⇒ *mode* ⇒ *'a* × *int*
where *may-o-create s pid dir dentry m* ≡
let
error = *resultValue s* (*security-path-mknod s dir dentry* (*nat m*) *0*)
in if error ≠ *0 then* (*s*,*error*)
else
let
s' = *funcState s* (*security-inode-create s* (*get-inode s* (*d-inode* (*p-dentry dir*))) *dentry m*);
retval = *resultValue s* (*security-inode-create s* (*get-inode s* (*d-inode*

(*p-dentry dir*))) *dentry m*)
in (*s'*,*retval*)

definition *filename-create* :: *int* ⇒ *string* ⇒ *path* ⇒*nat* ⇒*dentry option*
where *filename-create dfd name path lookup-flags* ≡ *Some*(*SOME x*:: *dentry* . *True*)

definition *getname pathname* ≡ *SOME x*::*string* . *True*

definition *user-path-create* :: *int* ⇒ *string* ⇒ *path* ⇒*nat* ⇒ *dentry option*
where *user-path-create dfd pathname path lookup-flags* ≡
let name = *getname pathname in*
filename-create dfd name path lookup-flags

definition *do-mknodat* :: *'a* ⇒ *process-id* ⇒ *int* ⇒ *string* ⇒ *mode* ⇒ *nat* ⇒ *'a* × *int*
where *do-mknodat s pid dfd filename m dev* ≡
let
path = *SOME x*:: *path* .*True* ;
lookup-flags = *0*;
dentry = (*the* (*user-path-create dfd filename path lookup-flags*));
error = *resultValue s* (*security-path-mknod s path dentry* (*nat m*) *dev*)
in if error ≠ *0 then*
(*s*,*error*)
else
(*s*,*0*)

typedecl *bpf-type*
definition *current-umask* :: *'a* ⇒ *int*
where *current-umask s* ≡ *umask* (*fs* (*current-process s*))

definition *bpf-obj-do-pin* :: *'a* ⇒ *process-id* ⇒ *string* ⇒ *string* ⇒ *bpf-type* ⇒ *'a* × *int*
where *bpf-obj-do-pin s pid pathname raw type'* ≡
let
path = *SOME x*:: *path* .*True* ;
mode = *bitOR S-IFREG* ((*bitOR S-IRUSR S-IWUSR*) *AND* (*NOT current-umask s*)) ;
dentry = *SOME x*::*dentry* . *True*;
ret = *resultValue s* (*security-path-mknod s path dentry* (*nat mode*) *0*)
in if ret ≠ *0 then*
(*s*,*ret*)
else
(*s*,*0*)

definition *unix-mknod* :: *'a* ⇒ *process-id* ⇒*string* ⇒ *mode* ⇒*path* ⇒ *'a* × *int*
where *unix-mknod s pid sun-path m res* ≡
let

path = SOME x:: path .True ;
dentry = SOME x::dentry . True;
ret = resultValue s (security-path-mknod s path dentry (nat m) 0)
in if ret ≠ 0 then
(s,ret)
else
(s,0)

29.9.3 kernel action of security$_{p}ath_{m}kdir$

definition *lookup-one-len :: 'a ⇒string ⇒ dentry ⇒int ⇒ dentry*
where *lookup-one-len s name base len' ≡ SOME x :: dentry . True*

definition *cachefiles-get-directory :: 'a ⇒ process-id ⇒ cachefiles-cache ⇒ dentry ⇒ string ⇒ 'a × dentry option*
where *cachefiles-get-directory s pid cache' dir dirname ≡*
let
path = SOME x:: path .True ;
path = path (| p-mnt := cc-mnt cache', p-dentry := dir|);
subdir = lookup-one-len s dirname dir (int (length(dirname)));
ret = resultValue s (security-path-mkdir s path subdir 448)
in if ret < 0 then (s,None)
else (s,Some subdir)

definition *SB-POSIXACL ≡ 1<<16*
definition *IS-FLG' inode flg ≡ (int(s-flags (i-sb inode))) AND flg*

definition *IS-POSIXACL :: inode ⇒ int*
where *IS-POSIXACL inode ≡ IS-FLG' inode SB-POSIXACL*

definition *do-mkdirat :: 'a ⇒ process-id ⇒ int ⇒ string ⇒ mode ⇒ 'a × int*
where *do-mkdirat s pid dfd pathname m ≡*
let
path = SOME x:: path .True ;
dentry = SOME x::dentry . True;
inode = get-inode s (d-inode (p-dentry path));
mode = if ((IS-POSIXACL inode) = 0)
then (m AND (NOT (current-umask s)))
else (m);
ret = resultValue s (security-path-mkdir s path dentry (nat mode))
in (s,ret)

29.9.4 kernel action of security$_{p}ath_{r}mdir$

definition *do-rmdir :: 'a ⇒ process-id ⇒ int ⇒ string ⇒ 'a × int*
where *do-rmdir s pid dir dentry ≡*
let
path = SOME x:: path .True ;
dentry = SOME x::dentry . True;

ret = *resultValue s* (*security-path-rmdir s path dentry*)
in (*s*,*ret*)

typedecl *fscache-why-object-killed*
type-synonym *fswhyok* = *fscache-why-object-killed*

29.9.5 kernel action of security$_{path_unlink}$

definition *cachefiles-bury-object* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *cachefiles-cache* $\Rightarrow$ *cachefiles-object*

$\Rightarrow$ *dentry* $\Rightarrow$ *dentry* $\Rightarrow$ *bool* $\Rightarrow$ *fswhyok* $\Rightarrow 'a \times$ *int*
where *cachefiles-bury-object s pid cache′ object dir rep preemptive why* $\equiv$
if ¬ (*d-is-dir rep*) *then*
let
path = *SOME x*:: *path* .*True* ;
path = *path* (| *p-mnt* := *cc-mnt cache′*, *p-dentry* := *dir*|);
ret = *resultValue s* (*security-path-unlink s path rep*)
in if ret < *0 then* (*s*,*ret*) *else* (*s*,*0*)
else
let
path = *SOME x*:: *path* .*True* ;
path-to-graveyard = *SOME x*:: *path* .*True* ;
path = *path* (| *p-mnt* := *cc-mnt cache′*, *p-dentry* := *dir*|);
path-to-graveyard = *path-to-graveyard* (| *p-mnt* := *cc-mnt cache′*, *p-dentry* := *graveyard cache′*|);
nbuffer = *SOME x*::*string* .*True*;
grave = *lookup-one-len s nbuffer* (*graveyard cache′*) (*int*(*length*(*nbuffer*)));
ret = *resultValue s* (*security-path-rename s path rep path-to-graveyard grave 0*)
in if ret < *0 then* (*s*,*ret*) *else* (*s*,*0*)

definition *do-unlinkat* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *int* $\Rightarrow$ *string* $\Rightarrow 'a \times$ *int*
where *do-unlinkat s pid dfd name*$\equiv$
let
path = *SOME x*:: *path* .*True* ;
dentry = *SOME x*::*dentry* . *True*;
ret = *resultValue s* (*security-path-unlink s path dentry*)
in (*s*,*ret*)

29.9.6 kernel action of security$_{path_symlink}$

definition *do-symlinkat* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *string* $\Rightarrow$ *int* $\Rightarrow$ *string* $\Rightarrow 'a \times$ *int*
where *do-symlinkat s pid oldname newdfd newname*$\equiv$
let
path = *SOME x*:: *path* .*True* ;
lookup-flags = *0*;
dentry = *user-path-create newdfd newname path lookup-flags*;
ret = *resultValue s* (*security-path-symlink s path* (*the dentry*) *oldname*)
in (*s*,*ret*)

29.9.7 kernel action of security$_{path_link}$

definition *do-linkat* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *int* $\Rightarrow$ *string* $\Rightarrow$ *int* $\Rightarrow$ *string* $\Rightarrow$ *int* $\Rightarrow$ $'a \times$ *int*
where *do-linkat s pid olddfd oldname newdfd newname flgs* $\equiv$
let
oldpath = SOME x:: path .True ;
newpath = SOME x:: path .True ;
path = p-dentry oldpath;
dentry = SOME x::dentry . True;
how = 0;
how = if (flgs AND AT-EMPTY-PATH) $\neq$ *0 then LOOKUP-EMPTY*
else how;
how = if (flgs AND AT-SYMLINK-FOLLOW) $\neq$ *0 then bitOR how LOOKUP-FOLLOW*
else how;
lookup-flags = (how AND LOOKUP-REVAL);
new-dentry = (the(user-path-create newdfd newname newpath (nat lookup-flags)));
ret = resultValue s (security-path-link s path newpath new-dentry)
in (s,ret)

29.9.8 kernel action of security$_{path_rename}$

definition *do-renameat2* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *int* $\Rightarrow$ *string* $\Rightarrow$ *int* $\Rightarrow$ *string* $\Rightarrow$ *nat* $\Rightarrow$ $'a \times$ *int*
where *do-renameat2 s pid olddfd oldname newdfd newname flgs* $\equiv$
let
old-path = SOME x:: path .True ;
new-path = SOME x:: path .True ;
old-dentry = SOME x::dentry . True;
new-dentry = SOME x::dentry . True;
ret = resultValue s (security-path-rename s old-path old-dentry new-path new-dentry flgs)
in (s,ret)

29.9.9 kernel action of security$_{path_truncate}$

definition *handle-truncate* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Files* $\Rightarrow$ $'a \times$ *int*
where *handle-truncate s pid filp*$\equiv$
let
path = f-path filp;
ret = resultValue s (security-path-truncate s path)
in (s,ret)

definition *vfs-truncate* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *path* $\Rightarrow$ *loff-t* $\Rightarrow$ $'a \times$ *int*

where *vfs-truncate s pid path length'* ≡
let
ret = *resultValue s* (*security-path-truncate s path*)
in (*s*,*ret*)

definition *FMODE-PATH* ≡ *0x4000*
definition *f--fget-light* :: *'a* ⇒ *nat* ⇒ *int* => *nat*
where*f--fget-light s fd mask'* ≡ *let files* = *files* (*current-process s*)
in (*nat*(*count files*))

definition *f--fdget* :: *'a* ⇒ *nat* ⇒ *nat*
where *f--fdget s fd* ≡ *f--fget-light s fd FMODE-PATH*

definition *do-sys-ftruncate* :: *'a* ⇒ *process-id* ⇒ *int* ⇒ *loff-t* ⇒ *int* ⇒ *'a* × *int*
where *do-sys-ftruncate s pid fd length' small*≡
let
f = *SOME x*:: *fd*. *True*;
files = *fdfile f*;
path = *f-path files*;
ret = *resultValue s* (*security-path-truncate s path*)
in (*s*,*ret*)

29.9.10 kernel action of security$_{p}ath_{c}hmod$

definition *chmod-common* :: *'a* ⇒ *process-id* ⇒ *path* ⇒ *mode* ⇒ *'a* × *int*
where *chmod-common s pid path mode'*≡
let
inode = *get-inode s* (*d-inode* (*p-dentry path*));
mode = *nat mode'*;
ret = *resultValue s* (*security-path-chmod s path mode*)
in (*s*,*ret*)

29.9.11 kernel action of security$_{p}ath_{c}hown$

definition *chown-common* :: *'a* ⇒ *process-id* ⇒ *path* ⇒ *uid-t* ⇒ *gid-t*⇒ *'a* × *int*
where *chown-common s pid path user group'*≡
let
inode = *get-inode s* (*d-inode* (*p-dentry path*));
uid = *make-kuid* (*current-user-ns s*) *user*;
gid = *make-kgid* (*current-user-ns s*) *group'*;
ret = *resultValue s* (*security-path-chown s path uid gid*)
in (*s*,*ret*)

29.9.12 kernel action of security$_{p}ath_{c}hroot$

definition *ksys-chroot* :: *'a* ⇒ *process-id* ⇒ *string* ⇒ *'a* × *int*

where *ksys-chroot s pid filename* ≡
let
path = *SOME x*:: *path .True* ;
ret = *resultValue s* (*security-path-chroot s path*)
in (*s*,*ret*)

29.10 inode

29.10.1 kernel action of security$_{i}node_{c}reate$

definition *cachefiles-check-cache-dir* :: $'a \Rightarrow$ *process-id* ⇒ *cachefiles-cache* ⇒ *dentry* ⇒ $'a \times$ *int*
where *cachefiles-check-cache-dir s pid cache′ root* ≡
let
ret = *resultValue s* (*security-inode-mkdir s* (*the*(*d-backing-inode s root*)) *root 0*)
in if ret < *0 then* (*s*,*ret*)
else
let
ret = *resultValue s* (*security-inode-create s* (*the*(*d-backing-inode s root*)) *root 0*)
in if ret < *0 then* (*s*, *ret*)
else (*s*,*0*)

definition *vfs-create* :: $'a \Rightarrow$ *process-id* ⇒ *inode* => *dentry* => *mode* => *bool* => $'a \times$ *int*
where *vfs-create s pid dir dentry m want-excl* ≡
let mode = *m AND S-IALLUGO*;
mode = *bitOR mode S-IFREG*;
ret = *resultValue s* (*security-inode-create s dir dentry mode*)
in (*s*,*ret*)

definition *vfs-mkobj* :: $'a \Rightarrow$*process-id* ⇒ *dentry* => *mode* => $'a \times$ *int*
where *vfs-mkobj s pid dentry m* ≡
let
dir = *get-inode s* (*d-inode* (*get-dentry s* (*d-parent dentry*)));
mode = *m AND S-IALLUGO*;
mode = *bitOR mode S-IFREG*;
ret = *resultValue s* (*security-inode-create s dir dentry mode*)
in (*s*,*ret*)

29.10.2 kernel action of security$_{i}node_{l}ink$

definition *vfs-link* :: $'a \Rightarrow$ *process-id* ⇒ *dentry* ⇒ *inode* ⇒ *dentry* ⇒ *inode* ⇒ $'a \times$ *int*
where *vfs-link s pid old-dentry dir new-dentry delegated-inode* ≡
let
ret = *resultValue s* (*security-inode-link s old-dentry dir new-dentry*)
in (*s*,*ret*)

29.10.3 kernel action of security$_{i}node_{u}nlink$

definition *vfs-unlink* :: $'a$ ⇒*process-id* => *inode* ⇒ *dentry* ⇒ *inode* => $'a$ × *int*
  where *vfs-unlink s pid dir dentry delegated-inode* ≡
    let
      ret = *resultValue s* (*security-inode-unlink s dir dentry*)
    in (*s*,*ret*)

29.10.4 kernel action of security$_{i}node_{s}ymlink$

definition *vfs-symlink* :: $'a$ ⇒*process-id* ⇒ *inode* ⇒ *dentry* ⇒ *string* => $'a$ × *int*
  where *vfs-symlink s pid dir dentry oldname* ≡
    let
      ret = *resultValue s* (*security-inode-symlink s dir dentry oldname*)
    in (*s*,*ret*)

29.10.5 kernel action of security$_{i}node_{m}kdir$

definition *vfs-mkdir* :: $'a$ ⇒*process-id* ⇒ *inode* ⇒ *dentry* ⇒ *mode* ⇒ $'a$ × *int*
  where *vfs-mkdir s pid dir dentry m* ≡
    let
      ret = *resultValue s* (*security-inode-mkdir s dir dentry m*)
    in (*s*,*ret*)

29.10.6 kernel action of security$_{i}node_{r}mdir$

definition *vfs-rmdir* :: $'a$ ⇒*process-id* ⇒ *inode* ⇒ *dentry* => $'a$ × *int*
  where *vfs-rmdir s pid dir dentry* ≡
    let
      ret = *resultValue s* (*security-inode-rmdir s dir dentry*)
    in (*s*,*ret*)

29.10.7 kernel action of security$_{i}node_{m}knod$

definition *vfs-mknod* :: $'a$ ⇒*process-id* ⇒ *inode* ⇒ *dentry* ⇒ *mode* ⇒ *dev-t* ⇒ $'a$ × *int*
  where *vfs-mknod s pid dir dentry m dev* ≡
    let
      ret = *resultValue s* (*security-inode-mknod s dir dentry m dev*)
    in (*s*,*ret*)

29.10.8 kernel action of security$_{i}node_{r}ename$

definition *vfs-rename* :: $'a$ ⇒ *process-id* ⇒ *inode* ⇒ *dentry*
      ⇒ *inode* ⇒ *dentry* ⇒ *inode* ⇒ *nat* ⇒ $'a$ × *int*

where *vfs-rename s pid old-dir old-dentry new-dir new-dentry delegated-inode flgs* $\equiv$
let
ret = *resultValue s* (*security-inode-rename s old-dir old-dentry new-dir new-dentry flgs*)
in (*s*,*ret*)

29.10.9 kernel action of security$_{i}node_{r}eadlink$

definition *vfs-get-link* :: $'a \Rightarrow$*process-id* $\Rightarrow$ *dentry* $\Rightarrow$ *delayed-call* $\Rightarrow$ $'a \times$ *int*
where *vfs-get-link s pid dentry done* $\equiv$
let
ret = *resultValue s* (*security-inode-readlink s dentry*)
in (*s*,*ret*)

definition *do-readlinkat* :: $'a \Rightarrow$*process-id* $\Rightarrow$*int*$\Rightarrow$*string* $\Rightarrow$*string*$\Rightarrow$*int* $\Rightarrow$ $'a \times$ *int*
where *do-readlinkat s pid dfd pathname buf bufsize* $\equiv$
let
path = *SOME x*:: *path*. *True*;
dentry = *p-dentry path*;
ret = *resultValue s* (*security-inode-readlink s dentry*)
in (*s*,*ret*)

29.10.10 kernel action of security$_{i}node_{f}ollow_{l}ink$

definition *get-link* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *nameidata* $\Rightarrow$ $'a \times$ *int*
where *get-link s pid nd* $\equiv$
let
depth = *depth nd* − *1*;
last = *stack nd* ! *depth*;
dentry = *p-dentry*(*saved-link last*);
inode = *link-inode nd*;
n = (*int*(*nd-flags nd*)) *AND LOOKUP-RCU*;
rcu = *if n* $\neq$ *0 then True else False*;
ret = *resultValue s* (*security-inode-follow-link s dentry inode rcu*)
in (*s*,*ret*)

29.10.11 kernel action of security$_{i}node_{p}ermission$

definition *inode-permission* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *inode* $\Rightarrow$ *int* $\Rightarrow$ $'a \times$ *int*
where *inode-permission s pid inode mask'* $\equiv$
let
ret = *resultValue s* (*security-inode-permission s inode mask'*)
in (*s*,*ret*)

29.10.12 kernel action of security$_{i}node_{s}etattrsecurity_{i}node_{n}eed_{k}illpriv$

definition *notify-change* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *dentry* $\Rightarrow$ *iattr* $\Rightarrow$ *inode* $\Rightarrow$ $'a \times$ *int*
  where *notify-change s pid dentry attr′ delegated-inode* $\equiv$
    let
      inode = *get-inode s* (*d-inode dentry*);
      ia-valid = *ia-valid attr′*;
      ret = (*if* (*int ia-valid AND ATTR-KILL-PRIV*) = *0 then*
          resultValue s (*security-inode-setattr s dentry attr′*)
        else
          resultValue s (*security-inode-need-killpriv s dentry*))
    in (*s*,*ret*)

definition *current-time* :: *inode* $\Rightarrow$ *timespec64*
  where *current-time i* $\equiv$ *SOME x*:: *timespec64*. *True*

29.10.13 kernel action of security$_{i}node_{s}etattr$

definition *fat-ioctl-set-attributes* :: $'a \Rightarrow$*process-id* $\Rightarrow$*Files*$\Rightarrow$ $'a \times$ *int*
  where *fat-ioctl-set-attributes s pid f* $\equiv$
    let
      dentry = *p-dentry*(*f-path f*);
      inode = *file-inode f*;
      is-dir = *S-ISDIR* (*i-mode inode*);
      ia = *SOME x*:: *iattr* . *True*;
      sbi = *SOME x*:: *msdos-sb-info* .*True*;
      ia-valid′ = *nat*(*bitOR ATTR-MODE ATTR-CTIME*);
      attr′ = *SOME x*::*char*. *True*;
      ia-mode′ = *if is-dir then fat-make-mode sbi attr′* (*nat S-IRWXUGO*)
          else fat-make-mode sbi attr′ (*nat* ((*bitOR* (*bitOR S-IRUGO*
S-IWUGO)
              (*i-mode inode AND S-IXUGO*)))) ;
      ia = *ia*⦇ *ia-valid* := *ia-valid′*,*ia-ctime* :=*current-time inode* ⦈;
      ret = *resultValue s* (*security-inode-setattr s dentry ia*)
    in (*s*,*ret*)

29.10.14 kernel action of security$_{i}node_{g}etattr$

definition *vfs-getattr* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *path* $\Rightarrow$ $'a \times$ *int*
  where *vfs-getattr s pid path* $\equiv$
    let
      ret = *resultValue s* (*security-inode-getattr s path*)
    in (*s*,*ret*)

29.10.15 kernel action of security$_{i}node_{s}etxattr$

definition *vfs-setxattr* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *dentry* $\Rightarrow$ *xattr* $\Rightarrow$ *string* $\Rightarrow$ *nat* $\Rightarrow$
nat $\Rightarrow$ $'a \times$ *int*

where *vfs-setxattr s pid dentry name value size′ flgs* ≡
let
ret = resultValue s (security-inode-setxattr s dentry name value size′ flgs)
in (s,ret)

definition *vfs-setxattr-noperm* :: *′a ⇒ process-id ⇒ dentry ⇒ xattr ⇒ Void ⇒ nat ⇒ nat ⇒ ′a × int*
where *vfs-setxattr-noperm s pid dentry name value size′ flgs* ≡
let
inode = get-inode s (d-inode dentry);
f = int(i-opflags inode) AND IOP-XATTR ;
value′ = SOME v. String v = value;
s′ = funcState s (security-inode-post-setxattr s dentry name value′ size′ flgs)
in if f ≠ 0 then (s′,0)
else
let
suffix′ = SOME x::xattr . True;
s′ = funcState s (security-inode-setsecurity s inode suffix′ value size′ flgs);
ret = resultValue s (security-inode-setsecurity s inode suffix′ value size′ flgs)
in (s′,ret)

29.10.16 kernel action of security$_{i}node_{g}etxattr$

definition *vfs-getxattr* :: *′a ⇒ process-id ⇒ dentry ⇒ xattr ⇒ string ⇒ nat ⇒ ′a × int*
where *vfs-getxattr s pid dentry name value size′* ≡
let
ret = resultValue s (security-inode-getxattr s dentry name)
in (s,ret)

29.10.17 kernel action of security$_{i}node_{l}istxattr$

definition *vfs-listxattr* :: *′a ⇒process-id ⇒ dentry ⇒ string ⇒ nat ⇒ ′a × int*
where *vfs-listxattr s pid dentry value size′* ≡
let
inode = get-inode s (d-inodeid dentry);
ret = resultValue s (security-inode-listxattr s dentry)
in if (ret ≠ 0) then (s,ret)
else
let
ret = resultValue s(security-inode-listsecurity s inode (String value) size′)
in (s,ret)

29.10.18 kernel action of security$_{i}node_{r}emovexattr$

definition *vfs-removexattr* :: *'a ⇒process-id ⇒dentry ⇒xattr ⇒ 'a × int*
where *vfs-removexattr s pid dentry name ≡*
let
ret = resultValue s (security-inode-removexattr s dentry name)
in (s,ret)

29.10.19 kernel action of security$_{i}node_{n}eed_{k}illpriv$

definition *dentry-needs-remove-privs* :: *'a ⇒process-id ⇒dentry ⇒ 'a × int*
where *dentry-needs-remove-privs s pid dentry ≡*
let
ret = resultValue s (security-inode-need-killpriv s dentry)
in (s,ret)

definition *setattr-prepare* :: *'a ⇒ process-id ⇒ dentry ⇒ iattr ⇒ 'a × int*
where *setattr-prepare s pid dentry attr' ≡*
let
ret = resultValue s (security-inode-killpriv s dentry)
in (s,ret)

29.10.20 kernel action of security$_{i}node_{g}etsecurity$

definition *xattr-getsecurity* :: *'a ⇒ process-id ⇒ inode*
⇒ xattr ⇒ string ⇒ int ⇒ 'a × int
where *xattr-getsecurity s pid inode name value size'≡*
let
buffer = [];
ret = resultValue s (security-inode-getsecurity s inode name (String buffer) True)
in (s,ret)

29.10.21 kernel action of security$_{i}node_{s}etsecurity$

definition *kernfs-node-setsecdata* :: *kernfs-iattrs ⇒ string ⇒nat ⇒int*
where*kernfs-node-setsecdata ka value len'≡ 0*

definition *kernfs-security-xattr-set* :: *'a ⇒ process-id ⇒ inode ⇒ xattr*
⇒ string ⇒ nat⇒int⇒ 'a × int
where *kernfs-security-xattr-set s pid inode suffix' value size' flgs≡*
let
secdata = [];
attrs = SOME x::kernfs-iattrs .True;
ret = resultValue s (security-inode-setsecurity s inode suffix' (String value) size' flgs);
s' = funcState s (security-inode-setsecurity s inode suffix' (String value) size' flgs)
in if (ret ≠ 0) then (s',ret)
else

let
ret = resultValue s (security-inode-getsecctx s inode (secdata) 0)
in if(ret ≠ 0) then (s,ret)
else let
error = kernfs-node-setsecdata attrs secdata 0;
s′ = funcState s (security-release-secctx s secdata 0)
in (s,0)

29.10.22 kernel action of security$_i$*node$_l$istsecurity*

definition *nfs4-listxattr-nfs4-label* :: *′a ⇒process-id ⇒inode ⇒string ⇒ int⇒ ′a × int*
where *nfs4-listxattr-nfs4-label s pid inode name size′* ≡
let
ret = resultValue s (security-inode-listsecurity s inode (String name) size′)
in (s,ret)

definition *sockfs-listxattr* :: *′a ⇒ process-id ⇒ dentry ⇒ string ⇒ int ⇒ ′a × int*
where *sockfs-listxattr s pid dentry buffer size′*≡
let
inode = get-inode s (d-inodeid dentry);
ret = resultValue s (security-inode-listsecurity s inode (String buffer) size′)
in (s,ret)

29.10.23 kernel action of security$_i$*node$_g$etsecid*

definition *audit-copy-inode* :: *′a ⇒ process-id ⇒ audit-names ⇒ dentry ⇒ inode ⇒ ′a × unit*
where *audit-copy-inode s pid name dentry inode* ≡
let
s′ = funcState s (security-inode-getsecid s inode (osid name))
in (s′,())

definition *ima-match-rules* :: *′a ⇒ process-id ⇒ inode ⇒ ′a × bool*
where *ima-match-rules s pid inode* ≡
let
osid = SOME x:: u32 . True;
s′ = funcState s (security-inode-getsecid s inode osid)
in (s′,True)

29.10.24 kernel action of security$_i$*node$_c$opy$_u$p*

definition *ovl-get-tmpfile* :: *′a ⇒ process-id ⇒ ovl-copy-up-ctx ⇒ ′a × int*
where *ovl-get-tmpfile s pid c* ≡
let
dentry = copy-dentry c;
new-creds = None;

ret = *resultValue s* (*security-inode-copy-up s dentry new-creds*)
in (*s*,*ret*)

29.10.25 kernel action of $\textbf{security}_{i}node_{c}opy_{u}p_{x}attr$

definition *ovl-copy-xattr* :: *'a* ⇒ *process-id* ⇒ *dentry* ⇒ *dentry* ⇒ *'a* × *int*
where *ovl-copy-xattr s pid old new* ≡
let
name = *SOME x*:: *xattr* . *True*;
ret = *resultValue s* (*security-inode-copy-up-xattr s name*)
in (*s*,*ret*)

29.11 ipc

definition *ipcperms* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *nat* ⇒*'a* × *int*
where *ipcperms s pid ipcp flg* ≡
let retval = *resultValue s* (*security-ipc-permission s ipcp flg*)
in (*s*,*retval*)

definition *audit-ipc-obj* :: *'a* ⇒ *process-id*⇒*kern-ipc-perm* ⇒*'a* × *unit*
where *audit-ipc-obj s pid ipcp* ≡
let retval = *resultU* (*security-ipc-getsecid s ipcp 0*)
in (*s*,())

definition *load-msg* :: *'a* ⇒ *process-id* ⇒ *msg-msg* ⇒ *'a* × *msg-msg option*
where *load-msg s pid msg* ≡
let retval = *resultValue s* (*security-msg-msg-alloc s msg*)
in if retval = *0*
then (*snd*(*the-run-state*(*security-msg-msg-alloc s msg*) *s*), *Some msg*)
else (*s*,*None*)

definition *free-msg* :: *'a* ⇒ *process-id* ⇒ *msg-msg* ⇒ *'a* × *unit*
where *free-msg s pid msg* ≡ (*snd*(*the-run-state*(*security-msg-msg-free s msg*) *s*), ())

definition *newque* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒*'a* × *int*
where *newque s pid msq* ≡
let retval = *resultValue s* (*security-msg-queue-alloc s msq*)
in if retval = *0*
then (*snd*(*the-run-state*(*security-msg-queue-alloc s msq*) *s*), *id msq*)
else (*s*,*retval*)

definition *msg-rcu-free* :: *'a* ⇒ *process-id*⇒*kern-ipc-perm* ⇒*'a* × *unit*
where *msg-rcu-free s pid msq* ≡
(*snd*(*the-run-state*(*security-msg-queue-free s msq*) *s*), ())

definition *ksys-msgget* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *int* ⇒*'a* × *int*
where *ksys-msgget s pid msq msqflg*≡
let retval = *resultValue s* (*security-msg-queue-associate s msq msqflg*)
in (*s*,*retval*)

definition *msg-queue-msgctl* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *IPC-CMD* ⇒*'a* × *int*
  where *msg-queue-msgctl s pid msq cmd*≡
    let retval = resultValue s (security-msg-queue-msgctl s msq cmd)
    in (s,retval)

definition *do-msgsnd* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *msg-msg* ⇒ *int* ⇒*'a* × *int*
  where *do-msgsnd s pid msq msg msgflg*≡
    let retval = resultValue s (security-msg-queue-msgsnd s msq msg msgflg)
    in if retval ≠ 0 then (s,retval) else (s,0)

definition *msg-queue-msgrcv* :: *'a* ⇒ *process-id*⇒*kern-ipc-perm* ⇒*msg-msg*⇒*Task* ⇒*int*⇒*int* ⇒*'a* × *int*
  where *msg-queue-msgrcv s pid isp msq p long msqflg*≡
    let retval = resultValue s (security-msg-queue-msgrcv s isp msq p long msqflg)
    in (s,retval)

definition *newseg* :: *'a* ⇒ *process-id* ⇒ *ipc-namespace* ⇒ *ipc-params* ⇒*'a* × *int*
  where *newseg s pid ns params* ≡
    let shp = SOME x:: shmid-kernel . True;
      shm-perm = shm-perm shp;
      retval = resultValue s (security-shm-alloc s shm-perm)
    in if retval = 0
      then (snd(the-run-state(security-shm-alloc s shm-perm) s), 0)
      else (s,retval)

definition *shm-rcu-free* :: *'a* ⇒ *process-id*⇒*kern-ipc-perm* ⇒*'a* × *unit*
  where *shm-rcu-free s pid shmperm* ≡
    (snd(the-run-state(security-shm-free s shmperm) s), ())

definition *ksys-shmget* :: *'a* ⇒ *process-id*⇒*kern-ipc-perm* ⇒*int* ⇒*'a* × *int*
  where *ksys-shmget s pid shm shmflg*≡
    let retval = resultValue s (security-shm-associate s shm shmflg)
    in (s,retval)

definition *shm-msgctl* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *IPC-CMD* ⇒ *'a* × *int*
  where *shm-msgctl s pid shm cmd*≡
    let retval = resultValue s (security-shm-shmctl s shm cmd)
    in (s,retval)

definition *do-shmat* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *string* ⇒ *int* ⇒ *'a* × *int*
  where *do-shmat s pid shp shmaddr shmflg*≡

let
flgs = *MAP-SHARED*;
retval = *resultValue s* (*security-shm-shmat s shp shmaddr shmflg*)
in if retval ≠ *0 then* (*s*,*retval*)
else
let
file = *SOME x*:: *Files*. *True*;
prot = *if* (*shmflg AND SHM-RDONLY*) ≠ *0 then PROT-READ else* (*bitOR PROT-READ PROT-WRITE*);
prot = *if* (*shmflg AND SHM-EXEC*) ≠ *0 then bitOR prot PROT-EXEC else prot*;
retval = *resultValue s* (*security-mmap-file s file* (*nat prot*) *flgs*)
in (*s*,*retval*)

definition *newary* :: *'a* ⇒ *process-id* ⇒ *ipc-namespace* ⇒ *ipc-params* ⇒*'a* × *int*
where *newary s pid ns params* ≡
let sma = *SOME x*:: *sem-array* . *True*;
sem-perm = *sem-perm sma*;
retval = *resultValue s* (*security-sem-alloc s sem-perm*)
in if retval = *0*
then (*snd*(*the-run-state*(*security-sem-alloc s sem-perm*) *s*), *id sem-perm*)
else (*s*,*retval*)

definition *sem-rcu-free* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *'a* × *unit*
where *sem-rcu-free s pid semperm* ≡
(*snd*(*the-run-state*(*security-sem-free s semperm*) *s*), ())

definition *ksys-semget* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *int* ⇒*'a* × *int*
where *ksys-semget s pid sem semflg*≡
let retval = *resultValue s* (*security-sem-associate s sem semflg*)
in (*s*,*retval*)

definition *sem-msgctl* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *IPC-CMD* ⇒ *'a* × *int*
where *sem-msgctl s pid sem cmd*≡
let retval = *resultValue s* (*security-sem-semctl s sem cmd*)
in (*s*,*retval*)

definition *do-semtimedop* :: *'a* ⇒ *process-id* ⇒ *kern-ipc-perm* ⇒ *sembuf* ⇒ *nat*⇒*int* ⇒*'a* × *int*
where *do-semtimedop s pid sma sops nsops alter*≡
let retval = *resultValue s* (*security-sem-semop s sma sops nsops alter*)
in if retval ≠ *0 then* (*s*,*retval*) *else* (*s*,*0*)

29.12 d$_i$*nstantiate*

29.12.1 kernel action of security$_{di}$*nstantiate*

definition *d-instantiate* :: *'a* ⇒ *process-id* ⇒ *dentry* ⇒ *inode option* ⇒ *'a* × *unit*
where *d-instantiate s pid entry inode* ≡

if inode $\neq$ *None then*
let
  inode = *the inode*;
  retval = *resultValue s* (*security-d-instantiate s entry inode*);
  s' = *funcState s* (*security-d-instantiate s entry inode*)
in
  (s', *retval*)
else (*s*,())

definition *d-instantiate-new* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *dentry* $\Rightarrow$ *inode* $\Rightarrow 'a \times$ *unit*
  where *d-instantiate-new s pid entry inode* $\equiv$
let
  retval = *resultValue s* (*security-d-instantiate s entry inode*);
  s' = *funcState s* (*security-d-instantiate s entry inode*)
in
  (s', *retval*)

definition *d-instantiate-anon'* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *dentry* $\Rightarrow$ *inode* $\Rightarrow$ *bool* $\Rightarrow 'a$ $\times$ *dentry option*
  where *d-instantiate-anon' s pid entry inode disconnected* $\equiv$
let
  res = *SOME x*:: *dentry* . *True*;
  retval = *resultValue s* (*security-d-instantiate s entry inode*);
  s' = *funcState s* (*security-d-instantiate s entry inode*)
in
  (s', *Some res*)

definition *d-add* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *dentry* $\Rightarrow$ *inode option* $\Rightarrow 'a \times$ *unit*
  where *d-add s pid entry inode* $\equiv$
if inode $\neq$ *None then*
let
  inode = *the inode*;
  retval = *resultValue s* (*security-d-instantiate s entry inode*);
  s' = *funcState s* (*security-d-instantiate s entry inode*)
in
  (s', *retval*)
else (*s*,())

definition *d-splice-alias* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *inode* $\Rightarrow$ *dentry* $\Rightarrow 'a \times$ *dentry option*
  where *d-splice-alias s pid inode dentry* $\equiv$
let
  new = *SOME x*:: *dentry* . *True*;
  retval = *resultValue s* (*security-d-instantiate s dentry inode*);
  s' = *funcState s* (*security-d-instantiate s dentry inode*)
in
  (s', *Some new*)

definition *nfs-get-root* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *super-block* $\Rightarrow$ *nfs-fh* $\Rightarrow$ *string* $\Rightarrow 'a$ $\times$ *dentry option*

where *nfs-get-root s pid sb mntfh devname* $\equiv$
let
inode = *SOME x*:: *inode* . *True*;
ret = *SOME x*:: *dentry* . *True*;
retval = *resultValue s* (*security-d-instantiate s ret inode*);
s' = *funcState s* (*security-d-instantiate s ret inode*)
in
(s', *Some ret*)

29.13 file

29.13.1 kernel action of security$_{file_permission}$

definition *iterate-dir* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Files* $\Rightarrow 'a \times$ *int*
where *iterate-dir s pid file* $\equiv$
let retval = *resultValue s* (*security-file-permission s file MAY-READ*) ;
s' = *funcState s* (*security-file-permission s file MAY-READ*)
in (s',*retval*)

definition *vfs-fallocate* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Files* $\Rightarrow$ *int* $\Rightarrow$ *loff-t* $\Rightarrow$ *loff-t* $\Rightarrow 'a \times$ *int*
where *vfs-fallocate s pid file m offset len'* $\equiv$
let retval = *resultValue s* (*security-file-permission s file MAY-WRITE*)
in if retval $\neq$ *0 then*
(*s*,*retval*)
else
(*s*,*0*)

definition *rw-verify-area* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *int* $\Rightarrow$ *Files* $\Rightarrow$ *loff-t* $\Rightarrow$ *nat* $\Rightarrow 'a \times$ *int*
where *rw-verify-area s pid rw file ppos count'* $\equiv$
let
flgs = *if rw* = *KREAD then MAY-READ else MAY-WRITE*;
retval = *resultValue s* (*security-file-permission s file flgs*)
in
(*s*,*retval*)

definition *clone-verify-area* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Files* $\Rightarrow$ *loff-t* $\Rightarrow$ *nat* $\Rightarrow$ *bool* $\Rightarrow 'a \times$ *int*
where *clone-verify-area s pid file pos len' write* $\equiv$
let
flgs = *if write then MAY-READ else MAY-WRITE*;
retval = *resultValue s* (*security-file-permission s file flgs*)
in
(*s*,*retval*)

29.13.2 kernel action of security$_{file_alloc}$

definition *alloc-file*:: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Files* $\Rightarrow$ *Cred* $\Rightarrow 'a \times$ *Files option*
where *alloc-file s pid file c*$\equiv$

let retval = resultValue s (security-file-alloc s file);
s′ = funcState s (security-file-alloc s file)
in
if retval ≠ 0 then
(s,None)
else
(s′,Some file)

29.13.3 kernel action of security$_{f}ile_{f}ree$

definition *file-free*:: *′a ⇒ process-id ⇒ Files ⇒′a × unit*
where *file-free s pid file ≡ (funcState s (security-file-free s file) ,())*

29.13.4 kernel action of security$_{f}ile_{i}octl$

definition *do-ioctl* :: *′a ⇒ process-id ⇒ Files ⇒ IOC-DIR ⇒ nat ⇒′a × int*
where *do-ioctl s pid file cmd arg≡*
let retval = resultValue s (security-file-ioctl s file cmd arg);
s′ = funcState s (security-file-ioctl s file cmd arg)
in if retval ≠ 0 then (s′,retval)
else (s′,0)

definition *syscall-ioctl* :: *′a ⇒ process-id⇒nat ⇒IOC-DIR ⇒nat ⇒′a × int*
where *syscall-ioctl s pid fd cmd arg≡*
let
file = SOME x:: Files. True;
retval = resultValue s (security-file-ioctl s file cmd arg)
in (s,retval)

definition *ksys-ioctl* :: *′a ⇒ process-id ⇒ nat ⇒ IOC-DIR ⇒ nat ⇒′a × int*
where *ksys-ioctl s pid fd cmd arg≡*
let
file = SOME x:: Files. True;
retval = resultValue s (security-file-ioctl s file cmd arg)
in (s,retval)

29.13.5 kernel action of security$_{m}map_{f}ile$

definition *vm-mmap-pgoff* :: *′a ⇒ process-id ⇒ Files ⇒ nat*
⇒ nat ⇒ nat ⇒ nat ⇒ nat ⇒′a × int
where *vm-mmap-pgoff s pid file addr len′ prot flag pgoff≡*
let
retval = resultValue s (security-mmap-file s file prot flag)
in (s,retval)

29.13.6 kernel action of security$_{m}map_{a}ddr$

definition *do-sys-vm86* :: *′a ⇒ process-id ⇒′a × int*
where *do-sys-vm86 s pid ≡*

let
retval = *resultValue s* (*security-mmap-addr s 0*)
in
(*s*,*retval*)

definition *get-unmapped-area* :: *'a* ⇒ *process-id* ⇒ *Files* ⇒ *nat* ⇒*'a* × *int*
where *get-unmapped-area s pid file addr* ≡
let
retval = *resultValue s* (*security-mmap-addr s addr*)
in if retval ≠ *0 then*
(*s*,*retval*)
else (*s*,*addr*)

definition *validate-mmap-request* :: *'a* ⇒ *process-id* ⇒ *Files* ⇒ *nat* ⇒*'a* × *int*
where *validate-mmap-request s pid file addr* ≡
let
retval = *resultValue s* (*security-mmap-addr s addr*)
in if retval < *0 then*
(*s*,*retval*)
else
(*s*,*0*)

29.13.7 kernel action of security$_{f}ile_{m}protect$

definition *do-mprotect-pkey* :: *'a* ⇒ *process-id* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒*'a* × *int*
where *do-mprotect-pkey s pid start len' prot pkey* ≡
let
vma = *SOME x*:: *vm-area-struct* .*True*;
rier = (*int*(*personality* (*current-process s*)) *AND READ-IMPLIES-EXEC*) ≠ *0* ∧
(((*int prot*) *AND PROT-READ*) ≠ *0*);
prot = (*int prot*) *AND* (*NOT* (*bitOR PROT-GROWSDOWN PROT-GROWSUP*));
reqprot = (*nat prot*);
prot = *if rier* ∧ (*vm-flags vma AND VM-MAYEXEC*) ≠ *0*
then bitOR prot PROT-EXEC
else prot;
retval = *resultValue s* (*security-file-mprotect s vma reqprot* (*nat prot*))
in (*s*,*retval*)

29.13.8 kernel action of security$_{f}ile_{l}ock$

definition *generic-setlease* :: *'a* ⇒ *process-id* ⇒ *Files* ⇒ *int* ⇒*'a* × *int*
where *generic-setlease s pid file arg* ≡
let retval = *resultValue s* (*security-file-lock s file* (*nat arg*))
in if retval ≠ *0 then* (*s*,*retval*) *else* (*s*,−*EINVAL*)

definition *syscall-lock* :: *'a* ⇒ *process-id* ⇒ *nat* ⇒ *nat* ⇒*'a* × *int*

where *syscall-lock s pid fd cmd* ≡
let file = *SOME x::Files. True*;
retval = *resultValue s* (*security-file-lock s file cmd*)
in (*s*,*retval*)

definition *do-lock-file-wait* :: *'a* ⇒ *process-id* ⇒ *Files* ⇒ *int* ⇒ *file-lock* ⇒*'a* × *int*
where *do-lock-file-wait s pid file cmd fl* ≡
let
arg = *of-char* (*fl-type fl*);
retval = *resultValue s* (*security-file-lock s file* (*nat arg*))
in if retval ≠ *0 then* (*s*,*retval*) *else* (*s*,*0*)

29.13.9 kernel action of security$_{file_fcntl}$

definition *file-fcntl* :: *'a* ⇒ *process-id* ⇒ *Files* ⇒ *nat* ⇒ *nat* ⇒*'a* × *int*
where *file-fcntl s pid file cmd arg*≡
let retval = *resultValue s* (*security-file-fcntl s file cmd arg*)
in if retval ≠ *0 then* (*s*,*retval*) *else* (*s*,*0*)

29.13.10 kernel action of security$_{file_set_fowner}$

definition *f-setown*:: *'a* ⇒ *process-id*⇒*Files* ⇒*'a* × *unit*
where *f-setown s pid file* ≡ (*snd*(*the-run-state*(*security-file-set-fowner s file*) *s*),())

29.13.11 kernel action of security$_{file_send_sigiotask}$

definition *file-send-sigiotask* :: *'a* ⇒ *process-id* ⇒ *Task* ⇒ *fown-struct* ⇒ *int* ⇒*'a* × *int*
where *file-send-sigiotask s pid t fown sig*≡
let retval = *resultValue s* (*security-file-send-sigiotask s t fown sig*)
in (*s*,*retval*)

29.13.12 kernel action of security$_{file_receive}$

definition *file-receive* :: *'a* ⇒ *process-id* ⇒ *Files* ⇒*'a* × *int*
where *file-receive s pid f*≡
let retval = *resultValue s* (*security-file-receive s f*)
in (*s*,*retval*)

29.13.13 kernel action of security$_{file_open}$

definition *do-dentry-open* :: *'a* ⇒ *process-id* ⇒ *Files* ⇒*'a* × *int*
where *do-dentry-open s pid f*≡
let retval = *resultValue s* (*security-file-open s f*)
in (*s*,*retval*)

29.14 net

29.14.1 kernel action of security$_n$*etlink*$_s$*end*

definition *netlink-sendmsg* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *socket* $\Rightarrow$ *msghdr* $\Rightarrow$ *nat* $\Rightarrow 'a \times$ *int*
where *netlink-sendmsg s pid sock msg len'*$\equiv$
let
sk' = the(sk sock);
skb = SOME x:: sk-buff .True;
retval = resultValue s (security-netlink-send s sk' skb)
in (s,retval)

29.14.2 kernel action of security$_i$*smaclabel*

definition *nfs4-xattr-set-nfs4-label* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *xattr*
$\Rightarrow$ *inode* $\Rightarrow$ *string* $\Rightarrow 'a \times$ *int*
where *nfs4-xattr-set-nfs4-label s pid key' inode buf*$\equiv$
let
retval = resultValue s (security-ismaclabel s key')
in if retval $\neq$ *0 then (s,0)*
else (s,−EOPNOTSUPP)

definition *nfs4-xattr-get-nfs4-label* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *xattr*
$\Rightarrow$ *inode* $\Rightarrow$ *string* $\Rightarrow 'a \times$ *int*
where *nfs4-xattr-get-nfs4-label s pid key' inode buf*$\equiv$
let
retval = resultValue s (security-ismaclabel s key')
in if retval $\neq$ *0 then (s,0)*
else (s,−EOPNOTSUPP)

29.15 secid$_t$*o*$_s$*ecctx*

29.15.1 kernel action of security$_s$*ecid*$_t$*o*$_s$*ecctx*

definition *scm-passec* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*socket* $\Rightarrow$ *msghdr* $\Rightarrow$ *scm-cookie* $\Rightarrow 'a \times$ *unit*
where *scm-passec s pid sock msg scm*$\equiv$
let
secdata = SOME x:: string .True;
seclen' = length(secdata);
secid = scm-secid scm;
retval = resultValue s (security-secid-to-secctx s secid secdata seclen')
in if retval = 0 then
let s' = funcState s (security-release-secctx s secdata seclen')
in (s',())
else (s,())

definition *audit-receive-msg* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *sk-buff* $\Rightarrow$ *nlmsghdr* $\Rightarrow 'a \times$ *int*
where *audit-receive-msg s pid skb nlh* $\equiv$
let

msg-type = nlmsg-type nlh
in
if msg-type = nat(AUDIT-SIGNAL-INFO) then let
secdata = '''';
seclen' = 0;
secid' = nat audit-sig-sid;
retval = resultValue s (security-secid-to-secctx s secid' secdata seclen')

in (s,retval)
else (s,0)

definition *audit-log-name :: 'a ⇒ process-id ⇒ audit-names ⇒'a × unit*
where *audit-log-name s pid n≡*
let
secdata = '''';
seclen' = length(secdata);
secid = osid n;
retval = resultValue s (security-secid-to-secctx s secid secdata seclen')
in if retval = 0 then
let s' = funcState s (security-release-secctx s secdata seclen')
in (s',())
else (s,())

definition *audit-log-task-context :: 'a ⇒ process-id ⇒ audit-buffer ⇒'a × int*
where *audit-log-task-context s pid skb ≡*
let
secdata = '''';
seclen' = 0;
secid' = SOME x:: nat .True;
retval = resultValue s (security-secid-to-secctx s secid' secdata seclen')
in if retval = 0 then
let s' = funcState s (security-release-secctx s secdata seclen')
in (s',0)
else (s,0)

definition *audit-log-pid-context :: 'a ⇒ process-id ⇒ u32 ⇒'a × int*
where *audit-log-pid-context s pid sid ≡*
if sid ≠ 0 then
let
secdata = '''';
seclen' = 0;
retval = resultValue s (security-secid-to-secctx s sid secdata seclen')
in if retval = 0 then
let s' = funcState s (security-release-secctx s secdata seclen')
in (s',0)
else (s,1)
else (s,0)

definition *show-special* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *audit-context* $\Rightarrow'a \times$ *unit*
where *show-special s pid context*$\equiv$
let
secdata = $''''$;
seclen' = *length*(*secdata*);
secid = *audit-context-ipc-osid* (*ipc context*);
retval = *resultValue s* (*security-secid-to-secctx s secid secdata seclen'*)
in if retval = *0 then*
let s' = *funcState s* (*security-release-secctx s secdata seclen'*)
in (*s'*,())
else (*s*,())

definition *ctnetlink-dump-secctx* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *sk-buff* $\Rightarrow$ *nf-conn* $\Rightarrow'a \times$ *int*
where *ctnetlink-dump-secctx s pid skb ct* $\equiv$
let
secdata = $''''$;
seclen' = *0*;
sid = *nf-secmark ct*;
retval = *resultValue s* (*security-secid-to-secctx s sid secdata seclen'*)
in if retval $\neq$ *0 then* (*s*,*0*)
else
let s' = *funcState s* (*security-release-secctx s secdata seclen'*)
in (*s'*,$-$*1*)

definition *ctnetlink-secctx-size* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *nf-conn* $\Rightarrow'a \times$ *int*
where *ctnetlink-secctx-size s pid ct* $\equiv$
let
secdata = $''''$;
seclen' = *0*;
sid = *nf-secmark ct*;
retval = *resultValue s* (*security-secid-to-secctx s sid secdata seclen'*)
in if retval $\neq$ *0 then* (*s*,*0*)
else (*s*,$-$*1*)

definition *ct-show-secctx* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*seq-file* $\Rightarrow$*nf-conn* $\Rightarrow'a \times$ *unit*
where *ct-show-secctx s pid seqfile ct*$\equiv$
let
secdata = *SOME x*::*string* .*True*;
seclen' = *length*(*secdata*);
secid = *nf-secmark ct*;
retval = *resultValue s* (*security-secid-to-secctx s secid secdata seclen'*)
in if retval = *0 then*
let s' = *funcState s* (*security-release-secctx s secdata seclen'*)
in (*s'*,())
else (*s*,())

definition *nfqnl-get-sk-secctx* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *sk-buff* $\Rightarrow$ *string* $\Rightarrow'a \times$ *int*
where *nfqnl-get-sk-secctx s pid skb secdata* $\equiv$

let
  seclen′ = 0;
  sid = secmark skb;
  retval = resultValue s (security-secid-to-secctx s sid secdata seclen′)
 in (s,int seclen′)

definition *netlbl-unlhsh-func3 :: ′a ⇒ process-id ⇒ u32 ⇒′a × int*
 where *netlbl-unlhsh-func3 s pid secid′ ≡*
  let
   secdata = ′′′′;
   secctx-len = SOME x:: u32. True;
   ret-val = SOME x:: int. True;
   sid = secid′;
   retval = resultValue s (security-secid-to-secctx s sid secdata secctx-len)
  in if retval ≠ 0 then (s,0)
   else
    let s′ = funcState s (security-release-secctx s secdata secctx-len)
    in (s′,ret-val)

definition *netlbl-unlabel-staticlist-gen :: ′a ⇒ process-id ⇒ u32 ⇒′a × int*
 where *netlbl-unlabel-staticlist-gen s pid secid′ ≡*
  let
   secctx = SOME x:: string. True;
   secctx-len = SOME x:: u32. True;
   sid = secid′;
   retval = resultValue s (security-secid-to-secctx s sid secctx secctx-len)
  in if retval ≠ 0 then (s,retval)
   else
    let s′ = funcState s (security-release-secctx s secctx secctx-len)
    in (s′,0)

definition *netlbl-audit-start-common :: ′a ⇒ process-id ⇒ int*
          ⇒ netlbl-audit ⇒′a × audit-buffer option
 where *netlbl-audit-start-common s pid type′ audit-info ≡*
  let
   buf = SOME x:: audit-buffer. True;
   secctx = SOME x:: string. True;
   secctx-len = SOME x:: u32. True;
   sid = netlbl-audit-secid audit-info;
  retval = resultValue s (security-secid-to-secctx s (nat sid) secctx secctx-len)

  in if sid ≠ 0 ∧ retval = 0 then
      let s′ = funcState s (security-release-secctx s secctx secctx-len)
       in (s′,Some buf)
      else (s,Some buf)

29.15.2 kernel action of security$_secctx_to_secid$

definition *set-security-override-from-ctx* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *Cred* $\Rightarrow$ *string* $\Rightarrow 'a \times$ *int*
where *set-security-override-from-ctx s pid new secctx* $\equiv$
let
secid = *SOME x*:: *u32* . *True*;
len = *length*(*secctx*);
retval = *resultValue s* (*security-secctx-to-secid s secctx len secid*)
in if retval < *0 then* (*s*,*retval*)
else (*s*, *snd*(*set-security-override s pid new secid*))

definition *netlbl-unlabel-staticadd* :: $'a \Rightarrow$ *process-id* $\Rightarrow 'a \times$ *int*
where *netlbl-unlabel-staticadd s pid* $\equiv$
let
secid = *SOME x*:: *u32* . *True*;
secctx = *SOME x*:: *string* . *True*;
len = *length*(*secctx*);
retval = *resultValue s* (*security-secctx-to-secid s secctx len secid*)
in if retval $\neq$ *0 then* (*s*,*retval*)
else (*s*, *0*)

29.15.3 kernel action of security$_release_secctx$

definition *kernfs-put* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *kernfs-node* $\Rightarrow 'a \times$ *unit*
where *kernfs-put s pid kn* $\equiv$
let
secdata =*ia-secdata* (*kn-iattr kn*);
seclen' = *ia-secdata-len* (*kn-iattr kn*);
s' = *funcState s* (*security-release-secctx s secdata seclen'*)
in (*s'*,())

definition *nfs4-label-release-security*:: $'a \Rightarrow$ *process-id* $\Rightarrow$ *nfs4-label option* $\Rightarrow 'a \times$ *unit*
where *nfs4-label-release-security s pid label'* $\equiv$ (*if label'* $\neq$ *None then*
let
secdata = *label* (*the label'*);
seclen' = *len* (*the label'*);
s' = *funcState s* (*security-release-secctx s secdata seclen'*)
in(*s'*,()) *else* (*s*,()))

29.15.4 kernel action of security$_inode_invalidate_secctx$

definition *inode-go-inval*:: $'a \Rightarrow$ *process-id* $\Rightarrow 'a \times$ *unit*
where *inode-go-inval s pid* $\equiv$
let
ip = *SOME x*:: *gfs2-inode* . *True*;
i = *i-inode ip*;

$s' = funcState\ s\ (\ security\text{-}inode\text{-}invalidate\text{-}secctx\ s\ i)$
$in(s',())$

29.15.5 kernel action of security$_inode_notifysecctx$

definition *kernfs-refresh-inode*:: $'a \Rightarrow process\text{-}id \Rightarrow kernfs\text{-}node \Rightarrow inode \Rightarrow 'a \times unit$
where *kernfs-refresh-inode s pid kn inode* $\equiv$
let
attrs = *kn-iattr kn*;
$s' = funcState\ s\ (security\text{-}inode\text{-}notifysecctx\ s\ inode\ (ia\text{-}secdata\ attrs)\ (ia\text{-}secdata\text{-}len\ attrs))$
$in(s',())$

definition *nfs-setsecurity*:: $'a \Rightarrow process\text{-}id \Rightarrow inode \Rightarrow nfs4\text{-}label \Rightarrow 'a \times unit$
where *nfs-setsecurity s pid inode label'* $\equiv$
let
secdata = *label label'*;
slen = *len label'*;
$s' = funcState\ s\ (\ security\text{-}inode\text{-}notifysecctx\ s\ inode\ (\ secdata)\ slen)$
$in(s',())$

29.15.6 kernel action of security$_inode_setsecctx$

definition *nfsd4-security-inode-setsecctx*:: $'a \Rightarrow process\text{-}id \Rightarrow svc\text{-}fh \Rightarrow xdr\text{-}netobj \Rightarrow u32 \Rightarrow 'a \times unit$
where *nfsd4-security-inode-setsecctx s pid resfh label' bmval* $\equiv$
let
d = *fh-dentry resfh*;
secdata = *xdr-data label'*;
slen = *xdr-len label'*;
$s' = funcState\ s\ (\ security\text{-}inode\text{-}setsecctx\ s\ d\ (\ secdata)\ slen)$
$in(s',())$

definition *nfsd4-set-nfs4-label*:: $'a \Rightarrow process\text{-}id \Rightarrow svc\text{-}fh \Rightarrow xdr\text{-}netobj \Rightarrow 'a \times int$
where *nfsd4-set-nfs4-label s pid resfh label'* $\equiv$
let
d = *fh-dentry resfh*;
secdata = *xdr-data label'*;
slen = *xdr-len label'*;
$s' = funcState\ s\ (\ security\text{-}inode\text{-}setsecctx\ s\ d\ (secdata)\ slen)$;
$retval = resultValue\ s\ (\ security\text{-}inode\text{-}setsecctx\ s\ d\ (\ secdata)\ slen)$
$in(s',retval)$

29.15.7 kernel action of security$_inode_getsecctx$

definition *nfsd4-encode-fattr*:: $'a \Rightarrow process\text{-}id \Rightarrow dentry \Rightarrow 'a \times int$
where *nfsd4-encode-fattr s pid dentry'* $\equiv$

let
  $d = $ *get-inode s* (*d-inodeid dentry'*);
  context = '''' ;
  slen = *SOME x*:: *int* . *True*;
  retval = *resultValue s* (*security-inode-getsecctx s d* (*context*) *slen*)
in(*s*,*retval*)

29.16 socket

29.16.1 kernel action of security$_unix_stream_connect$

definition *unix-stream-connect* :: *'a* ⇒ *process-id* ⇒ *socket* ⇒ *sockaddr* ⇒ *int* ⇒ *int* ⇒*'a* × *int*
  where *unix-stream-connect s pid sock uaddr addr-len flags'*≡
    let
     sk' = *the*(*sk sock*);
     other = *SOME x*:: *sock* . *True*;
     newsk = *SOME x*:: *sock* . *True*;
    retval = *resultValue s* (*security-unix-stream-connect s sk' other newsk*)
    in (*s*,*retval*)

29.16.2 kernel action of security$_unix_may_send$

definition *unix-dgram-connect* :: *'a* ⇒ *process-id*⇒*socket*⇒ *sockaddr* ⇒*int* ⇒*int* ⇒*'a* × *int*
  where *unix-dgram-connect s pid sock uaddr alen flags'*≡
    let
     sk' = *the*(*sk sock*);
     newsk = *get-socket s* (*sk-socket sk'*);
     other = *SOME x*:: *sock* . *True*;
     othersk = *get-socket s*(*sk-socket other*);
    retval = *resultValue s* (*security-unix-may-send s newsk othersk*)
    in (*s*,*retval*)

definition *unix-dgram-sendmsg* :: *'a* ⇒ *process-id* ⇒ *socket*⇒ *sockaddr* ⇒*int* ⇒*'a* × *int*
  where *unix-dgram-sendmsg s pid sock uaddr alen* ≡
    let
     sk' = *the*(*sk sock*);
     newsk = *get-socket s* (*sk-socket sk'*);
     other = *SOME x*:: *sock* . *True*;
     othersk = *get-socket s*(*sk-socket other*);
    retval = *resultValue s* (*security-unix-may-send s newsk othersk*)
    in (*s*,*retval*)

29.16.3 kernel action of security$_socket_createsecurity_socket_post_create$

definition *sock-alloc*:: *'a* ⇒ *socket option*
  where *sock-alloc s* ≡ *Some*(*SOME x*:: *socket*. *True*)

definition *sock-create-lite* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*Sk-Family* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *socket* $\Rightarrow 'a \times$ *int*
where *sock-create-lite s pid family type protocol' res* $\equiv$
let
retval = *resultValue s* (*security-socket-create s family type protocol' 1*)
in *if retval* $\neq$ *0 then*(*s*,*retval*)
else
let
sock = *sock-alloc s*
in
if sock = *None then* (*s*,−*ENOMEM*)
else
let
sock = *the sock*;
etval = *resultValue s* (*security-socket-post-create s sock family type protocol' 1*);
s' = *funcState s* (*security-socket-post-create s sock family type protocol' 1*)
in
(*s'*,*retval*)

definition *sock-create'* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *net* $\Rightarrow$ *Sk-Family* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *socket* $\Rightarrow$*int* $\Rightarrow 'a \times$ *int*
where *sock-create' s pid net' family type protocol' res kern* $\equiv$
let
retval = *resultValue s* (*security-socket-create s family type protocol' kern*)

in
if retval $\neq$ *0 then*
(*s*,*retval*)
else
let
sock = *sock-alloc s in*
if sock = *None then* (*s*,−*ENFILE*)
else
let
sock = *the sock*;
retval = *resultValue s* (*security-socket-post-create s sock family type protocol' 1*);
s' = *funcState s* (*security-socket-post-create s sock family type protocol' 1*)
in
(*s'*,*retval*)

29.16.4 kernel action of security$_{socket_socketpair}$

definition *sys-socketpair'* :: $'a \Rightarrow$ *process-id* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow 'a \times$ *int*

where *sys-socketpair′ s pid family type protocol′ usockvec* ≡
let
sock1 = *SOME x*:: *socket* . *True*;
sock2 = *SOME x*:: *socket* . *True*;
retval = *resultValue s* (*security-socket-socketpair s sock1 sock2*);
s′ = *funcState s* (*security-socket-socketpair s sock1 sock2*)
in
(*s′*,*retval*)

29.16.5 kernel action of security$_socket_bind$

definition *sys-bind′* :: *′a* ⇒ *process-id* ⇒ *int* ⇒ *sockaddr* ⇒ *int* ⇒*′a* × *int*
where *sys-bind′ s pid fd umyaddr addrlen* ≡
let
sock = *SOME x*:: *socket* . *True*;
address = *SOME x*:: *sockaddr* . *True*;
retval = *resultValue s* (*security-socket-bind s sock address addrlen*)
in
(*s*,*retval*)

29.16.6 kernel action of security$_socket_connect$

definition *sys-connect′* :: *′a* ⇒ *process-id* ⇒ *int* ⇒ *sockaddr* ⇒ *int* ⇒*′a* × *int*
where *sys-connect′ s pid fd uservaddr addrlen* ≡
let
sock = *SOME x*:: *socket* . *True*;
address = *SOME x*:: *sockaddr* . *True*;
retval = *resultValue s* (*security-socket-connect s sock address addrlen*)
in
(*s*,*retval*)

29.16.7 kernel action of security$_socket_listen$

definition *sys-listen′* :: *′a* ⇒ *process-id* ⇒ *int* ⇒ *int* ⇒*′a* × *int*
where *sys-listen′ s pid fd backlog* ≡
let
sock = *SOME x*:: *socket* . *True*;
retval = *resultValue s* (*security-socket-listen s sock backlog*)
in
(*s*,*retval*)

29.16.8 kernel action of security$_socket_accept$

definition *sys-accept4′* :: *′a* ⇒ *process-id* ⇒ *int* ⇒ *sockaddr* ⇒ *int* ⇒ *int* ⇒*′a* × *int*
where *sys-accept4′ s pid fd upeer-sockaddr upeer-addrlen flags′* ≡
let
sock = *SOME x*:: *socket* . *True*;

newsock = SOME x:: socket . True;
newsock = newsock (| *skt-type* := *skt-type sock* |);
retval = resultValue s (*security-socket-accept s sock newsock*)
in
(*s*,*retval*)

29.16.9 kernel action of security$_{s}ocket_{s}endmsg$

definition *iov-iter-count* :: *iov-iter* ⇒ *nat*
where*iov-iter-count i* ≡ *iov-count i*

definition *msg-data-left* :: *msghdr* ⇒ *nat*
where *msg-data-left msg* ≡ *iov-iter-count* (*msg-iter msg*)

definition *sock-sendmsg* :: *'a* ⇒ *process-id* ⇒ *socket* ⇒ *msghdr* ⇒*'a* × *int*
where *sock-sendmsg s pid sock msg* ≡
let
sock = SOME x:: socket . True;
l = msg-data-left msg;
retval = resultValue s (*security-socket-sendmsg s sock msg l*)
in
(*s*,*retval*)

29.16.10 kernel action of security$_{s}ocket_{r}ecvmsg$

definition *sock-recvmsg* :: *'a* ⇒ *process-id* ⇒ *socket* ⇒ *msghdr* ⇒ *int* ⇒ *'a* × *int*
where *sock-recvmsg s pid sock msg flags'* ≡
let
sock = SOME x:: socket . True;
l = msg-data-left msg;
retval = resultValue s (*security-socket-recvmsg s sock msg l flags'*)
in
(*s*,*retval*)

29.16.11 kernel action of security$_{s}ocket_{g}etsockname$

definition *sys-getsockname* :: *'a* ⇒ *process-id* ⇒ *int* ⇒ *sockaddr* ⇒ *int* ⇒*'a* × *int*
where *sys-getsockname s pid fd usockaddr usockaddr-len* ≡
let
sock = SOME x:: socket . True;
retval = resultValue s (*security-socket-getsockname s sock*)
in
(*s*,*retval*)

29.16.12 kernel action of security$_{s}ocket_{g}etpeername$

definition *sys-getpeername* :: *'a* ⇒ *process-id*⇒*int*⇒*sockaddr* ⇒ *int* ⇒*'a* × *int*
where *sys-getpeername s pid fd usockaddr usockaddr-len* ≡

let
sock = *SOME x*:: *socket* . *True*;
retval = *resultValue s* (*security-socket-getpeername s sock*)
in
(*s*,*retval*)

29.16.13 kernel action of security$_{s}ocket_{g}etsockopt$

definition *compat-sys-getsockopt′* :: *′a* ⇒ *process-id* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *string* ⇒*int*⇒*′a* × *int*
where *compat-sys-getsockopt′ s pid fd level′ optname optval optlen* ≡
let
sock = *SOME x*:: *socket* . *True*;
retval = *resultValue s* (*security-socket-getsockopt s sock level′ optname*)
in
(*s*,*retval*)

definition *sys-getsockopt′* :: *′a* ⇒ *process-id*⇒*int*⇒*int*⇒*int* ⇒*string* ⇒*int*⇒*′a* × *int*
where *sys-getsockopt′ s pid fd level′ optname optval optlen* ≡
let
sock = *SOME x*:: *socket* . *True*;
retval = *resultValue s* (*security-socket-getsockopt s sock level′ optname*)
in
(*s*,*retval*)

29.16.14 kernel action of security$_{s}ocket_{s}etsockopt$

definition *compat-sys-setsockopt′* :: *′a* ⇒ *process-id* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *string* ⇒ *int* ⇒*′a* × *int*
where *compat-sys-setsockopt′ s pid fd level′ optname optval optlen* ≡
let
sock = *SOME x*:: *socket* . *True*;
retval = *resultValue s* (*security-socket-setsockopt s sock level′ optname*)
in
(*s*,*retval*)

definition *sys-setsockopt′* :: *′a* ⇒ *process-id*⇒*int* ⇒ *int* ⇒ *int* ⇒ *string* ⇒ *int* ⇒*′a* × *int*
where *sys-setsockopt′ s pid fd level′ optname optval optlen* ≡
let
sock = *SOME x*:: *socket* . *True*;
retval = *resultValue s* (*security-socket-setsockopt s sock level′ optname*)
in
(*s*,*retval*)

29.16.15 kernel action of security$_{s}ocket_{s}hutdown$

definition *sys-shutdown′* :: *′a* ⇒ *process-id*⇒*int*⇒*int*⇒*′a* × *int*
where *sys-shutdown′ s pid fd how* ≡

let
sock = SOME x:: socket . True;
retval = resultValue s (security-socket-shutdown s sock how)
in
(s,retval)

29.16.16 kernel action of security$_{s}ock_{r}cv_{s}kb$

definition *sk-filter-trim-cap* :: *'a ⇒ process-id ⇒ sock ⇒ sk-buff ⇒ int⇒'a × int*
where *sk-filter-trim-cap s pid sk' skb cap ≡*
let
retval = resultValue s (security-sock-rcv-skb s sk' skb)
in
(s,retval)

29.16.17 kernel action of security$_{s}ocket_{g}etpeersec_{s}tream$

definition *sock-getsockopt* :: *'a ⇒ process-id ⇒ socket ⇒ int ⇒ int ⇒ string ⇒ int⇒'a × int*
where *sock-getsockopt s pid sock level' optname optval optlen ≡*
if optname = SO-PEERSEC then
let
sock = SOME x:: socket . True;
len = SOME x:: int . True;
retval = resultValue s (security-socket-getpeersec-stream s sock optval optlen len)
in
(s,retval)
else (s,0)

29.16.18 kernel action of security$_{s}ocket_{g}etpeersec_{d}gram$

definition *unix-get-peersec-dgram* :: *'a ⇒ process-id ⇒ socket ⇒ scm-cookie ⇒ 'a × unit*
where *unix-get-peersec-dgram s pid sock scm ≡*
let
sock = SOME x:: socket . True;
secid = scm-secid scm;
skb = None;
retval = resultValue s (security-socket-getpeersec-dgram s sock skb secid)
in
(s,())

definition *ip-cmsg-recv-security* :: *'a ⇒ process-id ⇒ msghdr ⇒ sk-buff ⇒'a × unit*
where *ip-cmsg-recv-security s pid msg skb ≡*
let
sock = SOME x:: socket . True;
secid =SOME x:: u32. True;

$skb = Some\ skb$;
$retval = resultValue\ s\ (\ security\text{-}socket\text{-}getpeersec\text{-}dgram\ s\ sock\ skb\ secid)$
$in\ if\ retval \neq 0\ then$
$(s,())$
$else$
let
$secdata = SOME\ x{::}\ string\ .\ True$;
$seclen = SOME\ x{::}\ u32.\ True$;
$retval = resultValue\ s\ (\ security\text{-}secid\text{-}to\text{-}secctx\ s\ secid\ secdata\ seclen)$
$in\ if\ retval \neq 0\ then$
$(s,())$
$else\ \ let\ s' = funcState\ s\ (\ security\text{-}release\text{-}secctx\ s\ \ secdata\ seclen)$
$in\ (s',())$

29.16.19 kernel action of security$_s k_a lloc$

definition $sk\text{-}prot\text{-}alloc :: 'a \Rightarrow process\text{-}id \Rightarrow proto \Rightarrow gfp\text{-}t \Rightarrow int \Rightarrow 'a \times sock\ option$
where $sk\text{-}prot\text{-}alloc\ s\ pid\ prot\ priority\ family \equiv$
let
$sk' = SOME\ x{::}\ sock\ option\ .\ True\ in$
$if\ sk' \neq None\ then$
let
$retval = resultValue\ s\ (\ security\text{-}sk\text{-}alloc\ s\ (the\ sk')\ family\ priority)$;
$s' = funcState\ s\ (\ security\text{-}sk\text{-}alloc\ s\ (the\ sk')\ family\ priority)$
in
$(s',\ sk')$
$else\ (s,None)$

29.16.20 kernel action of security$_s k_f ree$

definition $sk\text{-}prot\text{-}free :: 'a \Rightarrow process\text{-}id \Rightarrow proto \Rightarrow sock \Rightarrow 'a \times unit$
where $sk\text{-}prot\text{-}free\ s\ pid\ prot\ sk' \equiv$
let
$retval = resultValue\ s\ (\ security\text{-}sk\text{-}free\ s\ \ sk')$;
$s' = funcState\ s\ (\ security\text{-}sk\text{-}free\ s\ \ sk'\ )$
in
$(s',\ retval)$

29.16.21 kernel action of security$_s k_c lone$

definition $sk\text{-}clone :: 'a \Rightarrow process\text{-}id \Rightarrow sock \Rightarrow sock \Rightarrow 'a \times unit$
where $sk\text{-}clone\ s\ pid\ \ sk'\ newsk \equiv$
let
$retval = resultValue\ s\ (\ security\text{-}sk\text{-}clone\ s\ \ sk'\ newsk)$;

$s' = funcState\ s\ (\ security\text{-}sk\text{-}clone\ s\ \ sk'\ newsk)$
in
$(s',\ retval)$

29.16.22 kernel action of security$_sk_classify_flow$

definition *sk-classify-flow* :: $'a \Rightarrow process\text{-}id \Rightarrow sock \Rightarrow flowi \Rightarrow 'a \times unit$
where *sk-classify-flow s pid sk' fl* $\equiv$
let
$retval = resultValue\ s\ (\ security\text{-}sk\text{-}classify\text{-}flow\ s\ \ sk'\ fl);$
$s' = funcState\ s\ (\ security\text{-}sk\text{-}classify\text{-}flow\ s\ \ sk'\ fl)$
in
$(s',\ retval)$

29.16.23 kernel action of security$_req_classify_flow$

definition *req-classify-flow* :: $'a \Rightarrow process\text{-}id \Rightarrow request\text{-}sock \Rightarrow flowi \Rightarrow 'a \times unit$
where *req-classify-flow s pid sk' fl* $\equiv$
let
$retval = resultValue\ s\ (\ security\text{-}req\text{-}classify\text{-}flow\ s\ \ sk'\ fl);$
$s' = funcState\ s\ (\ security\text{-}req\text{-}classify\text{-}flow\ s\ \ sk'\ fl)$
in
$(s',\ retval)$

29.16.24 kernel action of security$_sock_graft$

definition *af-alg-accept* :: $'a \Rightarrow process\text{-}id \Rightarrow sock \Rightarrow socket \Rightarrow bool \Rightarrow 'a \times unit$
where *af-alg-accept s pid sk' newsock kern* $\equiv$
let
$sk2 = SOME\ x{::}sock\ .True;$
$retval = resultValue\ s\ (\ security\text{-}sock\text{-}graft\ s\ \ sk2\ newsock);$
$s' = funcState\ s\ (\ security\text{-}sock\text{-}graft\ s\ \ sk2\ newsock)$
in
$(s',\ retval)$

definition *sock-graft* :: $'a \Rightarrow process\text{-}id \Rightarrow sock \Rightarrow socket \Rightarrow 'a \times unit$
where *sock-graft s pid sk' parent'* $\equiv$
let
$retval = resultValue\ s\ (\ security\text{-}sock\text{-}graft\ s\ \ sk'\ parent');$
$s' = funcState\ s\ (\ security\text{-}sock\text{-}graft\ s\ \ sk'\ parent')$
in
$(s',\ retval)$

29.16.25 kernel action of security$_inet_conn_request$

definition *inet-conn-request* :: $'a \Rightarrow process\text{-}id \Rightarrow sock \Rightarrow sk\text{-}buff \Rightarrow request\text{-}sock \Rightarrow 'a \times int$
where *inet-conn-request s pid sk' skb req* $\equiv$
let
$retval = resultValue\ s\ (\ security\text{-}inet\text{-}conn\text{-}request\ s\ \ sk'\ skb\ req);$

$s' = funcState\ s\ (\ security\text{-}inet\text{-}conn\text{-}request\ s\ \ sk'\ skb\ req)$
in
$(s',\ retval)$

29.16.26 kernel action of security$_inet_csk_clone$

definition *inet-csk-clone-lock* :: $'a \Rightarrow process\text{-}id \Rightarrow sock \Rightarrow request\text{-}sock \Rightarrow gfp\text{-}t \Rightarrow 'a \times sock\ option$
where *inet-csk-clone-lock s pid sk′ req priority* $\equiv$
let
$newsk = SOME\ x{::}sock\ option.\ True$
in if $newsk \neq None$ *then*
let
$newsk = the\ newsk;$
$retval = resultValue\ s\ (\ security\text{-}inet\text{-}csk\text{-}clone\ s\ \ newsk\ req);$
$s' = funcState\ s\ (\ security\text{-}inet\text{-}csk\text{-}clone\ s\ \ newsk\ \ req)$
in
$(s',Some\ newsk)$
else $(s,None)$

29.16.27 kernel action of security$_inet_conn_established$

definition *tcp-finish-connect* :: $'a \Rightarrow process\text{-}id \Rightarrow sock \Rightarrow sk\text{-}buff \Rightarrow 'a \times unit$
where *tcp-finish-connect s pid sk′ skb* $\equiv$
let
$retval = resultValue\ s\ (\ security\text{-}inet\text{-}conn\text{-}established\ s\ \ sk'\ skb);$
$s' = funcState\ s\ (\ security\text{-}inet\text{-}conn\text{-}established\ s\ \ sk'\ skb)$
in
$(s',\ retval)$

definition *sctp-sf-do-5-1E-ca* :: $'a \Rightarrow process\text{-}id \Rightarrow sctp\text{-}endpoint \Rightarrow 'a \times unit$
where *sctp-sf-do-5-1E-ca s pid ep* $\equiv$
let
$chunk = SOME\ x{::}\ sctp\text{-}chunk\ .True;$
$skb = sctp\text{-}skb\ chunk;$
$sk' = sctp\text{-}ep\text{-}sk(sctp\text{-}base\ ep);$
$retval = resultValue\ s\ (\ security\text{-}inet\text{-}conn\text{-}established\ s\ \ sk'\ skb);$
$s' = funcState\ s\ (\ security\text{-}inet\text{-}conn\text{-}established\ s\ \ sk'\ skb)$
in
$(s',\ retval)$

29.16.28 kernel action of security$_secmark_relabel_packetsecurity_secmark_refcount_inc$

definition *checkentry-lsm* :: $'a \Rightarrow process\text{-}id \Rightarrow xt\text{-}secmark\text{-}target\text{-}info \Rightarrow 'a \times int$
where *checkentry-lsm s pid info* $\equiv$
let
$secid' = xt\text{-}secid\ info;$
$secctx = xt\text{-}secctx\ info;$
$err = resultValue\ s\ (\ security\text{-}secctx\text{-}to\text{-}secid\ s\ secctx\ 256\ secid'\ )$

in if err ≠ 0 then (s,err)
else let
retval = resultValue s (security-secmark-relabel-packet s secid');
s' = funcState s (security-secmark-relabel-packet s secid')
in if retval ≠ 0 then (s',retval)
else
let s'' = funcState s (security-secmark-refcount-inc s)
in (s'',0)

29.16.29 kernel action of security$_secmark_refcount_dec$

term *(int(of-char xt-mode))*
definition *secmark-tg-destroy :: 'a ⇒ process-id ⇒ 'a × unit*
where *secmark-tg-destroy s pid ≡*
if (int(of-char xt-mode)) = SECMARK-MODE-SEL then
let
s' = funcState s (security-secmark-refcount-dec s)
in
(s', ())
else
(s,())

29.17 key

29.17.1 kernel action of security$_key_alloc$

definition *key-alloc :: 'a ⇒ process-id ⇒ key-type ⇒ string ⇒ kuid-t ⇒ kgid-t*
⇒ Cred ⇒ nat ⇒'a × key option
where *key-alloc s pid ktype desc uid' gid' cred' flags' ≡*
let
key = SOME x:: key. True;
retval = resultValue s (security-key-alloc s key cred' flags');
s' = funcState s (security-key-alloc s key cred' flags')
in if retval < 0 then
(s,None)
else
(s', Some key)

29.17.2 kernel action of security$_key_free$

definition *key-free :: 'a ⇒ process-id⇒key ⇒'a × unit*
where *key-free s pid key' ≡*
let
retval = resultValue s (security-key-free s key');
s' = funcState s (security-key-free s key')
in
(s',retval)

29.17.3 kernel action of security$_k ey_p ermission$

definition *key-task-permission* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*key-ref-t* $\Rightarrow$ *Cred* $\Rightarrow$ *nat* $\Rightarrow 'a \times$ *int*
where *key-task-permission s pid key-ref cred' perm* $\equiv$
let
retval = *resultValue s* (*security-key-permission s key-ref cred' perm*)
in
(*s*,*retval*)

29.17.4 kernel action of security$_k ey_g etsecurity$

definition *key-ref-to-ptr* :: $'a \Rightarrow$ *key-ref-t* => *key option*
where *key-ref-to-ptr s key-ref* $\equiv$ ((*keys s*) *key-ref*)

definition *keyctl-get-security* :: $'a \Rightarrow$ *process-id*$\Rightarrow$*key-serial-t* $\Rightarrow$ *string* $\Rightarrow$*int* $\Rightarrow 'a \times$ *int*
where *keyctl-get-security s pid keyid' buffer buflen*$\equiv$
let
key-ref = *keyid'*;
key = *the*(*key-ref-to-ptr s key-ref*);
context = *''''*;
retval = *resultValue s* (*security-key-getsecurity s key context*)
in
(*s*,*retval*)

29.18 audit

29.18.1 kernel action of security$_a udit_r ule_i nit$

definition *audit-data-to-entry* :: $'a \Rightarrow$ *process-id* $\Rightarrow 'a \times$ *int*
where *audit-data-to-entry s pid* $\equiv$
let
f = *SOME x*:: *audit-field*. *True*;
ftype = *atype f*;
fop = *aop f*;
rule = *lsm-rule f*;
str = *SOME x*:: *string*. *True*;
retval = *resultValue s* (*security-audit-rule-init s ftype fop str rule*)
in
(*s*,*retval*)

definition *audit-dupe-lsm-field* :: $'a \Rightarrow$ *process-id* $\Rightarrow$*audit-field* $\Rightarrow$ *audit-field*$\Rightarrow 'a \times$ *int*
where *audit-dupe-lsm-field s pid df sf* $\equiv$
let
df = *df*(|*lsm-str* := *lsm-str sf*|);
ftype = *atype df*;

fop = aop df;
rule = lsm-rule df;
str = lsm-str sf;
retval = resultValue s (security-audit-rule-init s ftype fop str rule)
in if retval = (−EINVAL) then (s,0)
else
(s,retval)

29.18.2 kernel action of security$_a udit_r ule_k nown$

definition *update-lsm-rule* :: *'a ⇒ process-id ⇒ audit-krule⇒'a × int*
where *update-lsm-rule s pid r≡*
let
retval = resultValue s (security-audit-rule-known s r)
in
(s,retval)

29.18.3 kernel action of security$_a udit_r ule_f ree$

definition *audit-free-lsm-field* :: *'a ⇒ process-id ⇒ audit-field⇒'a × unit*
where *audit-free-lsm-field s pid f ≡*
if (atype f) = AUDIT-OBJ-LEV-HIGH then
let
retval = resultValue s (security-audit-rule-free s (lsm-rule f))
in
(s,retval)
else (s,())

29.18.4 kernel action of security$_a udit_r ule_m atch$

definition *audit-rule-match* :: *'a ⇒ process-id ⇒ int⇒'a × int*
where *audit-rule-match s pid sid ≡*
let
f = SOME x:: audit-field. True;
ftype = atype f;
fop = aop f;
rule = lsm-rule f;
ac = SOME x:: audit-context. True;
sid = nat sid;
retval = resultValue s (security-audit-rule-match s sid ftype fop rule ac)
in
(s,retval)

29.19 bpf

29.19.1 kernel action of security$_b pf$

definition *syscall-bpf* :: *'a ⇒ process-id ⇒ int ⇒ nat⇒'a × int*

where *syscall-bpf s pid cmd size'* $\equiv$
let
attr= SOME x:: bpf-attr . True;

retval = resultValue s (security-bpf s cmd attr size')
in if retval < 0 then
(s,retval)
else
(s,0)

29.19.2 kernel action of security$_bpf_map$

definition *bpf-map-new-fd :: 'a ⇒ process-id ⇒bpf-map ⇒ int⇒'a × int*
where *bpf-map-new-fd s pid map' flags'* $\equiv$
let
attr= SOME x:: bpf-attr . True;
flag = OPEN-FMODE flags';
retval = resultValue s (security-bpf-map s map' flag)
in if retval < 0 then
(s,retval)
else
(s,0)

29.19.3 kernel action of security$_bpf_prog$

definition *get-prog-inode :: 'a ⇒ process-id ⇒bpf-map ⇒ int⇒'a × bpf-prog option*
where *get-prog-inode s pid map' flags'* $\equiv$
let
prog= SOME x:: bpf-prog . True;
retval = resultValue s (security-bpf-prog s prog)
in if retval < 0 then
(s,None)
else
(s,Some prog)

definition *bpf-prog-new-fd :: 'a ⇒ process-id ⇒bpf-prog⇒'a × int*
where *bpf-prog-new-fd s pid prog* $\equiv$
let
retval = resultValue s (security-bpf-prog s prog)
in if retval < 0 then
(s,retval)
else
(s,0)

29.19.4 kernel action of security$_bpf_map_alloc$

definition *map-create :: 'a ⇒ process-id ⇒bpf-attr⇒'a × int*
where *map-create s pid attr'* $\equiv$
let

map= SOME x:: bpf-map . True;
retval = resultValue s (security-bpf-map-alloc s map);
s′ = funcState s (security-bpf-map-alloc s map)
in if retval < 0 then
(s,retval)
else
(s′,0)

29.19.5 kernel action of security$_bpf_prog_alloc$

definition *bpf-prog-load :: ′a ⇒ process-id ⇒bpf-attr⇒′a × int*
where *bpf-prog-load s pid attr′ ≡*
let
prog= SOME x:: bpf-prog . True;
aux′ = aux prog;
retval = resultValue s (security-bpf-prog-alloc s aux′);
s′ = funcState s (security-bpf-prog-alloc s aux′)
in if retval < 0 then
(s,retval)
else
(s′,0)

29.19.6 kernel action of security$_bpf_map_free$

definition *container-of-map :: work-struct ⇒ bpf-map*
where *container-of-map work′ ≡ SOME x . work x = work′*

definition *bpf-map-free-deferred :: ′a ⇒ process-id ⇒work-struct⇒′a × unit*
where *bpf-map-free-deferred s pid work′ ≡*
let
map= container-of-map work′;
s′ = funcState s (security-bpf-map-free s map)
in
(s′,())

29.19.7 kernel action of security$_bpf_prog_free$

definition *container-of-progfree :: rcu-head ⇒ bpf-prog-aux*
where *container-of-progfree rcu′ ≡ SOME x . rcu x = rcu′*

definition *bpf-prog-put-rcu :: ′a ⇒ process-id ⇒rcu-head⇒′a × unit*
where *bpf-prog-put-rcu s pid work′ ≡*
let
aux = container-of-progfree work′;
s′ = funcState s (security-bpf-prog-free s aux)
in
(s′,())

29.20 event pid

definition *getpid-from-sb-event* :: *Event-sb* ⇒ *process-id*
where *getpid-from-sb-event evt* ≡ (*case evt of*
Event-sb-copy-data pid sb ⇒ *pid*
| *Event-sb-remount pid p v* ⇒ *pid*
| *Event-sb-kern-mount pid f t name data* ⇒ *pid*
| *Event-sb-show-options pid sq sb* ⇒ *pid*
| *Event-sb-statfs pid d* ⇒*pid*
| *Event-sb-mount pid devname dirname t f p* ⇒ *pid*
| *Event-sb-umount pid m i* ⇒ *pid*
| *Event-sb-pivotroot pid* ⇒ *pid*
| *Event-set-mnt-opts pid n sb opt* ⇒ *pid*
| *Event-set-sb-security pid sb d info* ⇒*pid*
| *Event-sb-clone-mnt-opts pid sb d minfo* ⇒ *pid*
| *Event-sb-parse-opts-str pid string opts* ⇒ *pid*
)

definition *getpid-from-tsk-event* :: *Event-tsk* ⇒ *process-id*
where *getpid-from-tsk-event e* ≡ (*case e of*
(*Event-prepare-creds process-id*) ⇒ *process-id*
|(*Event-sys-setreuid process-id uid′ euid′*) ⇒ *process-id*
|(*Event-setpgid process-id i pgid*) ⇒ *process-id*
|(*Event-do-getpgid process-id pgid*) ⇒ *process-id*
|(*Event-getsid process-id sid*) ⇒ *process-id*
|(*Event-getsecid process-id p u*) ⇒ *process-id*
|(*Event-task-setnice process-id p nice*) ⇒ *process-id*
|(*Event-set-task-ioprio process-id p ioprio*) ⇒ *process-id*
|(*Event-get-task-ioprio process-id p*) ⇒ *process-id*
|(*Event-check-prlimit-permission process-id p i*) ⇒
process-id
|(*Event-do-prlimit process-id p i*) ⇒ *process-id*
|(*Event-task-setscheduler process-id p*) ⇒ *process-id*
|(*Event-task-getscheduler process-id p*) ⇒ *process-id*
|(*Event-task-movememory process-id p*) ⇒ *process-id*
|(*Event-task-kill process-id p siginfo i c*) ⇒ *process-id*
|(*Event-task-prctl process-id op arg2 arg3 arg4 arg5*) ⇒
process-id
)

definition *getpid-from-ptrace-event* :: *Event-Ptrace* ⇒ *process-id*
where *getpid-from-ptrace-event e* ≡ (*case e of*
Event-ptrace-access-check process-id Task m
⇒*process-id*
| *Event-ptrace-traceme process-id* => *process-id*)

definition *getpid-from-sys-event* :: *Event-sys* ⇒ *process-id*
where *getpid-from-sys-event e* ≡ (*case e of*
Event-smack-syslog process-id t ⇒*process-id* |
Event-prepare-binprm process-id bprm ⇒ *process-id*)

definition *getpid-from-kern-ipc-event* :: *Event-ipc* ⇒ *process-id*
where *getpid-from-kern-ipc-event e* ≡ (*case e of*
(*Event-ipc-permission process-id kern-ipc-perm flg*) ⇒ *process-id*
| (*Event-ipc-getsecid process-id kern-ipc-perm*) ⇒ *process-id*
| (*Event-msg-queue-associate process-id kern-ipc-perm flg*) ⇒*process-id*
| (*Event-msg-queue-msgctl process-id kern-ipc-perm flg*) ⇒*process-id*
| (*Event-msg-queue-msgsnd process-id kern-ipc-perm msg-msg flg*) ⇒*process-id*
| (*Event-msg-queue-msgrcv process-id kern-ipc-perm msg-msg p long msqflg*) ⇒*process-id*
| (*Event-shm-associate process-id kern-ipc-perm flg*) ⇒*process-id*
| (*Event-shm-shmctl process-id kern-ipc-perm flg*) ⇒*process-id*
| (*Event-shm-shmat process-id kern-ipc-perm string flg*) ⇒*process-id*

| (*Event-sem-associate process-id kern-ipc-perm flg*) ⇒*process-id*
| (*Event-sem-semctl process-id kern-ipc-perm flg*) ⇒ *process-id*
| (*Event-sem-semop process-id kern-ipc-perm sembuf nsops alter*) ⇒ *process-id*
)

definition *getpid-from-file-event* :: *Event-file* ⇒ *process-id*
where *getpid-from-file-event e* = (*case e of*
Event-do-ioctl process-id Files IOC-DIR arg ⇒ *process-id*
| *Event-syscall-ioctl process-id fd IOC-DIR arg* ⇒ *process-id*
| *Event-ksys-ioctl process-id fd IOC-DIR arg* ⇒ *process-id*
| *Event-vm-mmap-pgoff process-id file addr len′ prot flag pgoff* ⇒ *process-id*
| *Event-do-sys-vm86 process-id* ⇒ *process-id*
| *Event-get-unmapped-area process-id Files addr* ⇒ *process-id*
| *Event-validate-mmap-request process-id Files addr*⇒ *process-id*
| *Event-generic-setlease process-id Files arg* ⇒ *process-id*
| *Event-syscall-lock process-id fd cmd* ⇒ *process-id*
| *Event-do-lock-file-wait process-id Files cmd file-lock* ⇒ *process-id*
| *Event-file-fcntl process-id Files cmd arg* ⇒ *process-id*
| *Event-file-send-sigiotask process-id Task fown-struct sig* ⇒ *process-id*
| *Event-file-receive process-id Files* ⇒ *process-id*
| *Event-do-dentry-open process-id Files*⇒ *process-id*)

definition *getpid-from-key-evevt* :: *Event-Key* ⇒ *process-id*
where *getpid-from-key-evevt e* = (*case e of*
Event-key-permission process-id key-ref-t Cred prem ⇒ *process-id*
| *Event-key-getsecurity process-id key-serial-t buffer buflen* ⇒ *process-id*)

definition *getpid-from-aduit-evevt* :: *Event-audit* ⇒ *process-id*
where *getpid-from-aduit-evevt e* = (*case e of*
Event-audit-data-to-entry process-id ⇒ *process-id*
| *Event-audit-dupe-lsm-field process-id df sf* ⇒ *process-id*
| *Event-audit-rule-known process-id krule* ⇒ *process-id*
| *Event-audit-rule-match process-id sid* ⇒ *process-id*
| *Event-audit-rule-free process-id field* ⇒ *process-id*)

definition *getpid-from-socket-evevt* :: *Event-network-sock* ⇒ *process-id*
where *getpid-from-socket-evevt e* = (*case e of*
Event-unix-stream-connect process-id sock uaddr addr-len flags' ⇒ *process-id*
| *Event-unix-dgram-connect process-id sock uaddr alen flags'* ⇒ *process-id*

| *Event-unix-dgram-sendmsg process-id sock uaddr alen* ⇒ *process-id*

| *Event-sys-bind' process-id fd umyaddr addrlen* ⇒ *process-id*
| *Event-sys-connect' process-id fd uservaddr addrlen*⇒ *process-id*
| *Event-sock-sendmsg process-id sock msg* ⇒ *process-id*
| *Event-sock-recvmsg process-id sock msg flags'* ⇒ *process-id*
| *Event-sk-filter-trim-cap process-id sk' skb cap* ⇒ *process-id*
| *Event-sock-getsockopt process-id sock level' optname optval optlen* ⇒ *process-id*
| *Event-unix-get-peersec-dgram process-id sock scm* ⇒ *process-id*
)

definition *getpid-from-inode-evevt* :: *Event-inode* ⇒ *process-id*
where *getpid-from-inode-evevt e* = (*case e of*
Event-vfs-link process-id old dir new delegated-inode ⇒ *process-id*
| *Event-vfs-unlink process-id dir dentry delegated-inode* ⇒ *process-id*
| *Event-vfs-rmdir process-id inode dentry* ⇒ *process-id*
| *Event-vfs-rename process-id old-dir old-dentry new-dir new-dentry delegated-inode flgs* ⇒ *process-id*
| *Event-inode-permission process-id inode mask'* ⇒ *process-id*
| *Event-notify-change process-id dentry iattr inode* ⇒ *process-id*
| *Event-fat-ioctl-set-attributes process-id Files* ⇒ *process-id*
| *Event-vfs-getattr process-id path* ⇒ *process-id*
| *Event-vfs-setxattr process-id dentry xattr string size' flgs* ⇒ *process-id*
| *Event-vfs-getxattr process-id dentry xattr string size'* ⇒ *process-id*
| *Event-vfs-removexattr process-id dentry xattr* ⇒ *process-id*
| *Event-xattr-getsecurity process-id inode name value size'* ⇒ *process-id*

| *Event-nfs4-listxattr-nfs4-label process-id inode string size'* ⇒ *process-id*
| *Event-sockfs-listxattr process-id dentry string size'* ⇒ *process-id*
)

definition *getpid-from-istpro-evevt* :: *Event-d-instantiate-procattr* ⇒ *process-id*
where *getpid-from-istpro-evevt e* = (*case e of*
Event-d-instantiate pid entry inode ⇒ *pid*
| *Event-d-instantiate-new pid entry inode* ⇒ *pid*
| *Event-d-instantiate-anon' pid entry inode disconnected* ⇒ *pid*
| *Event-d-add pid entry inode* ⇒ *pid*
| *Event-d-splice-alias pid inode dentry* ⇒ *pid*
| *Event-nfs-get-root pid sb mntfh devname* ⇒ *pid*
| *Event-proc-pid-attr-read pid file buf count' ppos* ⇒ *pid*
| *Event-proc-pid-attr-write pid file buf count' ppos*⇒ *pid*)

definition *getpid-from-other-evevt* :: *Event-others* ⇒ *process-id*
where *getpid-from-other-evevt e* = (*case e of*
Evt-ismaclabel(*Event-nfs4-xattr-set-nfs4-label pid key' inode buf*) ⇒ *pid*
| *Evt-ismaclabel*(*Event-nfs4-xattr-get-nfs4-label pid key' inode buf*) ⇒ *pid*
| *Evt-id2secctx*(*Event-scm-passec pid sock msg scm*) ⇒ *pid*
| *Evt-id2secctx*(*Event-audit-receive-msg pid skb nlh*) ⇒ *pid*
| *Evt-id2secctx*(*Event-audit-log-name pid n*) ⇒ *pid*
| *Evt-id2secctx*(*Event-audit-log-task-context pid skb*) ⇒ *pid*
| *Evt-id2secctx*(*Event-audit-log-pid-context pid sid*)⇒ *pid*
| *Evt-id2secctx*(*Event-show-special pid context*) ⇒ *pid*
| *Evt-id2secctx*(*Event-ctnetlink-dump-secctx pid skb ct*) ⇒ *pid*
| *Evt-id2secctx*(*Event-ctnetlink-secctx-size pid ct*) ⇒ *pid*
| *Evt-id2secctx*(*Event-ct-show-secctx pid seqfile ct*) ⇒ *pid*
| *Evt-id2secctx*(*Event-nfqnl-get-sk-secctx pid skb secdata*)⇒ *pid*
| *Evt-id2secctx*(*Event-netlbl-unlhsh-func3 pid secid'*)⇒ *pid*
| *Evt-id2secctx*(*Event-netlbl-unlabel-staticlist-gen pid secid'*) ⇒ *pid*
| *Evt-id2secctx*(*Event-netlbl-audit-start-common pid type' audit-info*) ⇒ *pid*
| *Evt-secctxtoid*(*Event-set-security-override-from-ctx pid new secctx*) ⇒ *pid*
| *Evt-secctxtoid*(*Event-netlbl-unlabel-staticadd pid*) ⇒ *pid*
| *Evt-inodesecctx*(*Event-kernfs-refresh-inode pid kn inode*)⇒ *pid*
| *Evt-inodesecctx*(*Event-nfs-setsecurity pid inode label'*)⇒ *pid*
| *Evt-inodesecctx*(*Event-nfsd4-security-inode-setsecctx pid resfh label' bmval*) ⇒ *pid*
| *Evt-inodesecctx*(*Event-nfsd4-set-nfs4-label pid resfh label'*) ⇒ *pid*
| *Evt-inodesecctx*(*Event-nfsd4-encode-fattr pid dentry'*)⇒ *pid*
| *Evt-inodesecctx*(*Event-ovl-get-tmpfile pid c*)⇒ *pid*
| *Evt-inodesecctx*(*Event-ovl-copy-xattr pid old new*) ⇒ *pid*
| *Evt-inodesecctx*(*Event-ovl-create-or-link pid dentry inode attr' origin*) ⇒ *pid*)

datatype *Event* = *SbEvt Event-sb*
| *TskEvt Event-tsk*
| *PtraceEvt Event-Ptrace*

| *SysEvt Event-sys*
| *FileEvt Event-file*
| *KIpcEvt Event-ipc*
| *KeyEvt Event-Key*
| *AuditEvt Event-audit*
| *SocketEvt Event-network-sock*
| *InodeEvt Event-inode*

datatype *Event-manage-hooks* = *Event-alloc-file process-id Files Cred*
| *Event-file-free process-id Files*

definition *getpid-from-manage-hooks* :: *Event-manage-hooks* ⇒ *process-id option*
where *getpid-from-manage-hooks e* = (*case e of*
Event-alloc-file process-id Files Cred ⇒ *Some process-id*
| *Event-file-free process-id Files* ⇒ *Some process-id*
)

definition *exec-manage-event* :: ′*a* ⇒*Event-manage-hooks* ⇒ ′*a*
where *exec-manage-event s e* = (*case e of*
Event-alloc-file pid file c⇒ *fst*(*alloc-file s pid file c*)|
Event-file-free pid file ⇒ *fst*(*file-free s pid file*))

29.21 Instantiation and Its Proofs of IFS Model

definition *exec-event* :: ′*a* ⇒*Event* ⇒ ′*a*
where *exec-event s e* = (*case e of*
SbEvt (*Event-sb-copy-data pid sb*) ⇒ *fst*(*k-sb-copy-data s pid*) |
SbEvt (*Event-sb-remount pid p v*) ⇒ *fst*(*do-remount s p v*) |
SbEvt (*Event-sb-kern-mount pid t f name data*) ⇒ *fst*(*mount-fs s t f name data*)|
SbEvt (*Event-sb-show-options pid sq sb*) ⇒ *fst*(*show-sb-opts s pid sq sb*) |
SbEvt (*Event-sb-statfs pid d*) ⇒*fst*(*statfs-by-dentry s pid d*) |
SbEvt (*Event-sb-mount pid devname dirname t f p*) ⇒ *fst*(*do-mount s pid devname dirname t f p*) |
SbEvt (*Event-sb-umount pid m i*) ⇒ *fst* (*do-umount s pid m i*) |
SbEvt (*Event-sb-pivotroot pid*) ⇒ *fst* (*pivot-root s pid*) |
SbEvt (*Event-set-mnt-opts pid n sb opt*) ⇒ *fst*(*setup-security-options s pid n sb opt*) |
SbEvt (*Event-set-sb-security pid sb d info*) ⇒*fst*(*set-sb-security s pid sb d info*) |
SbEvt (*Event-sb-clone-mnt-opts pid sb d minfo*) ⇒ *fst*(*nfs-clone-sb-security s pid sb d minfo*) |
SbEvt (*Event-sb-parse-opts-str pid str opts*) ⇒*fst*(*parse-security-options s pid str opts*) |

TskEvt ((*Event-prepare-creds pid*)) ⇒ *fst*(*prepare-creds s pid*) |
TskEvt ((*Event-sys-setreuid pid kuid euid*′)) ⇒ *fst*(*sys-setreuid s pid kuid euid*′) |

TskEvt ((*Event-setpgid pid i pgid*)) ⇒ *fst*(*setpgid s pid i pgid*) |
TskEvt ((*Event-do-getpgid pid i*)) ⇒*fst*(*do-getpgid s pid i*) |
TskEvt ((*Event-getsid pid i*)) ⇒*fst*(*getsid s pid i*) |
TskEvt ((*Event-getsecid pid p u*)) ⇒*fst*(*getsecid s pid p u*) |
TskEvt ((*Event-task-setnice pid p i*)) ⇒*fst*(*task-setnice s pid p i*) |
TskEvt ((*Event-set-task-ioprio pid p i*)) ⇒*fst*(*set-task-ioprio s pid p i*) |
TskEvt ((*Event-get-task-ioprio pid p*)) ⇒ *fst*(*get-task-ioprio s pid p*) |
TskEvt ((*Event-check-prlimit-permission pid p i*)) ⇒*fst*(*check-prlimit-permission s pid p i*) |
TskEvt ((*Event-do-prlimit pid p i*)) ⇒*fst*(*do-prlimit s pid p i*) |
TskEvt ((*Event-task-setscheduler pid p*)) ⇒*fst*(*task-setscheduler s pid p*) |
TskEvt ((*Event-task-getscheduler pid p*)) ⇒*fst*(*task-getscheduler s pid p*) |
TskEvt ((*Event-task-movememory pid p*)) ⇒*fst*(*task-movememory s pid p*) |
TskEvt ((*Event-task-kill pid p sinfo i c*)) ⇒*fst*(*task-kill s pid p sinfo i c*) |
TskEvt ((*Event-task-prctl pid op arg2 arg3 arg4 arg5*))
⇒*fst*(*task-prctl s pid op arg2 arg3 arg4 arg5*) |

SysEvt(*Event-smack-syslog pid t*)⇒ *fst*(*check-syslog-permissions s pid t*)|
SysEvt(*Event-prepare-binprm pid bprm*) ⇒*fst*(*prepare-binprm s pid bprm*) |
PtraceEvt (*Event-ptrace-access-check pid p m*) ⇒ *fst*(*ptrace-may-access s pid p m*)|
PtraceEvt (*Event-ptrace-traceme pid*) ⇒ *fst*(*ptrace-traceme s pid*)|

FileEvt(*Event-do-ioctl pid file cmd arg*)⇒ *fst*(*do-ioctl s pid file cmd arg*)|
FileEvt(*Event-syscall-ioctl pid fd cmd arg*)⇒ *fst*(*syscall-ioctl s pid fd cmd arg*)|
FileEvt(*Event-ksys-ioctl pid fd cmd arg*)⇒ *fst*(*ksys-ioctl s pid fd cmd arg*)|
FileEvt(*Event-vm-mmap-pgoff pid file addr len′ prot flag pgoff*)
⇒ *fst*(*vm-mmap-pgoff s pid file addr len′ prot flag pgoff*)|
FileEvt(*Event-do-sys-vm86 pid*)⇒ *fst*(*do-sys-vm86 s pid*)|
FileEvt(*Event-get-unmapped-area pid file addr*)⇒ *fst*(*get-unmapped-area s pid file addr*)|
FileEvt(*Event-validate-mmap-request pid file addr*)⇒ *fst*(*validate-mmap-request s pid file addr*)|
FileEvt(*Event-generic-setlease pid file arg*)⇒ *fst*(*generic-setlease s pid file arg*)|
FileEvt(*Event-syscall-lock pid fd cmd*)⇒ *fst*(*syscall-lock s pid fd cmd*)|
FileEvt(*Event-do-lock-file-wait pid file cmd fl*)⇒ *fst*(*do-lock-file-wait s pid file cmd fl*)|
FileEvt(*Event-file-fcntl pid file cmd arg*)⇒ *fst*(*file-fcntl s pid file cmd arg*)|
FileEvt(*Event-file-send-sigiotask pid t fown sig*)⇒ *fst*(*file-send-sigiotask s pid t fown sig*)|
FileEvt(*Event-file-receive pid f*)⇒ *fst*(*file-receive s pid f*)|
FileEvt(*Event-do-dentry-open pid f*)⇒ *fst*(*do-dentry-open s pid f*) |

KIpcEvt((*Event-ipc-permission pid ipcp flg*))⇒ *fst*(*ipcperms s pid ipcp flg*) |
KIpcEvt((*Event-ipc-getsecid pid ipcp*))⇒ *fst*(*audit-ipc-obj s pid ipcp*) |

KIpcEvt((*Event-msg-queue-associate pid msq msqflg*))⇒*fst*(*ksys-msgget s pid msq msqflg*) |
KIpcEvt((*Event-msg-queue-msgctl pid msq cmd*))⇒*fst*(*msg-queue-msgctl s pid msq cmd*) |
KIpcEvt((*Event-msg-queue-msgsnd pid msq msg msgflg*))⇒*fst*(*do-msgsnd s pid msq msg msgflg*) |
KIpcEvt((*Event-msg-queue-msgrcv pid isp msq p long msqflg*))
⇒*fst*(*msg-queue-msgrcv s pid isp msq p long msqflg*) |
KIpcEvt((*Event-shm-associate pid shm shmflg*))⇒*fst*(*ksys-shmget s pid shm shmflg*) |
KIpcEvt((*Event-shm-shmctl pid shm cmd*))⇒*fst*(*shm-msgctl s pid shm cmd*) |
KIpcEvt((*Event-shm-shmat pid shp shmaddr shmflg*))⇒*fst*(*do-shmat s pid shp shmaddr shmflg*) |
KIpcEvt((*Event-sem-associate pid sem semflg*))⇒*fst*(*ksys-semget s pid sem semflg*) |
KIpcEvt((*Event-sem-semctl pid sem cmd*))⇒ *fst*(*sem-msgctl s pid sem cmd*) |
KIpcEvt((*Event-sem-semop pid sma sops nsops alter*))
⇒ *fst*(*do-semtimedop s pid sma sops nsops alter*) |
KeyEvt(*Event-key-permission pid key-ref cred′ perm*)
⇒ *fst*(*key-task-permission s pid key-ref cred′ perm*)|
KeyEvt(*Event-key-getsecurity pid keyid′ buffer buflen*)
⇒ *fst*(*keyctl-get-security s pid keyid′ buffer buflen*)|

AuditEvt(*Event-audit-data-to-entry pid*) ⇒ *fst*(*audit-data-to-entry s pid*) |
AuditEvt(*Event-audit-dupe-lsm-field pid df sf*) ⇒ *fst*(*audit-dupe-lsm-field s pid df sf*) |
AuditEvt(*Event-audit-rule-known pid krule*) ⇒ *fst*(*update-lsm-rule s pid krule*) |
AuditEvt(*Event-audit-rule-match pid sid*) ⇒ *fst*(*audit-rule-match s pid sid*) |
AuditEvt(*Event-audit-rule-free pid f*) ⇒ *fst*(*audit-free-lsm-field s pid f*) |

SocketEvt(*Event-unix-stream-connect pid sock uaddr addr-len flags′*)
⇒ *fst*(*unix-stream-connect s pid sock uaddr addr-len flags′*)|
SocketEvt(*Event-unix-dgram-connect pid sock uaddr alen flags′*)
⇒ *fst*(*unix-dgram-connect s pid sock uaddr alen flags′*) |
SocketEvt(*Event-unix-dgram-sendmsg pid sock uaddr alen*)
⇒ *fst*(*unix-dgram-sendmsg s pid sock uaddr alen*)|
SocketEvt(*Event-sys-bind′ pid fd umyaddr addrlen*)⇒ *fst*(*sys-bind′ s pid fd umyaddr addrlen*)|
SocketEvt(*Event-sys-connect′ pid fd uservaddr addrlen*)⇒ *fst*(*sys-connect′ s pid fd uservaddr addrlen*)|
SocketEvt(*Event-sock-sendmsg pid sock msg*)⇒ *fst*(*sock-sendmsg s pid sock msg*)|
SocketEvt(*Event-sock-recvmsg pid sock msg flags′*)⇒ *fst*(*sock-recvmsg s pid sock msg flags′*) |
SocketEvt(*Event-sk-filter-trim-cap pid sk′ skb cap*)⇒ *fst*(*sk-filter-trim-cap s*

pid sk′ skb cap) |
SocketEvt(*Event-sock-getsockopt pid sock level′ optname optval optlen*)
⇒ *fst*(*sock-getsockopt s pid sock level′ optname optval optlen*) |
SocketEvt(*Event-unix-get-peersec-dgram pid sock scm*)⇒ *fst*(*unix-get-peersec-dgram s pid sock scm*) |

InodeEvt(*Event-vfs-link pid old-dentry dir new-dentry delegated-inode*)
⇒ *fst*(*vfs-link s pid old-dentry dir new-dentry delegated-inode*)|
InodeEvt(*Event-vfs-unlink pid dir dentry delegated-inode*)
⇒ *fst*(*vfs-unlink s pid dir dentry delegated-inode*) |
InodeEvt(*Event-vfs-rmdir pid dir dentry*)⇒ *fst*(*vfs-rmdir s pid dir dentry*) |
InodeEvt(*Event-vfs-rename pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*)
⇒ *fst*(*vfs-rename s pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*) |
InodeEvt(*Event-inode-permission pid inode mask′*)⇒ *fst*(*inode-permission s pid inode mask′*) |
InodeEvt(*Event-notify-change pid dentry attr′ delegated-inode*)
⇒ *fst*(*notify-change s pid dentry attr′ delegated-inode*) |
InodeEvt(*Event-fat-ioctl-set-attributes pid f*)⇒ *fst*(*fat-ioctl-set-attributes s pid f*) |
InodeEvt(*Event-vfs-getattr pid path*)⇒ *fst*(*vfs-getattr s pid path*) |
InodeEvt(*Event-vfs-setxattr pid dentry name value size′ flgs*)
⇒ *fst*(*vfs-setxattr s pid dentry name value size′ flgs*) |
InodeEvt(*Event-vfs-getxattr pid dentry name value size′*)
⇒ *fst*(*vfs-getxattr s pid dentry name value size′*) |
InodeEvt(*Event-vfs-removexattr pid dentry name*)
⇒ *fst*(*vfs-removexattr s pid dentry name*) |
InodeEvt(*Event-xattr-getsecurity pid inode name value size′*)
⇒ *fst*(*xattr-getsecurity s pid inode name value size′*) |
InodeEvt(*Event-nfs4-listxattr-nfs4-label pid inode name size′*)
⇒ *fst*(*nfs4-listxattr-nfs4-label s pid inode name size′*)|
InodeEvt(*Event-sockfs-listxattr pid dentry buffer size′*)
⇒ *fst*(*sockfs-listxattr s pid dentry buffer size′*)
)

definition *observe* :: *′a* ⇒ *nat* ⇒ *Obj set* (**infixl** *55*)
where *observe s subj* ≡ {*obj*. *obj* ∈ *Obj s* ∧
READ ∈ *access-rules* (*subj-label s subj*) (*obj-label s obj*)}

definition *alter* :: *′a* ⇒ *nat* ⇒ *Obj set* (**infixl** *56*)
where*alter s subj* ≡ {*obj* . *obj* ∈ *Obj s* ∧
WRITE ∈ *access-rules* (*subj-label s subj*) (*obj-label s obj*)}

primrec *domain-of-event* :: *Event* ⇒ *process-id option* **where**
domain-of-event (*SbEvt e*) = *Some* (*getpid-from-sb-event e*)|
domain-of-event (*TskEvt e*) = *Some* (*getpid-from-tsk-event e*) |
domain-of-event (*PtraceEvt e*) = *Some* (*getpid-from-ptrace-event e*)|
domain-of-event (*SysEvt e*) = *Some* (*getpid-from-sys-event e*) |
domain-of-event (*FileEvt e*) = *Some* (*getpid-from-file-event e*) |
domain-of-event (*KIpcEvt e*) = *Some* (*getpid-from-kern-ipc-event e*) |
domain-of-event (*KeyEvt e*) = *Some* (*getpid-from-key-evevt e*) |
domain-of-event (*AuditEvt e*) = *Some* (*getpid-from-aduit-evevt e*) |
domain-of-event (*SocketEvt e*) = *Some* (*getpid-from-socket-evevt e*) |
domain-of-event (*InodeEvt e*) = *Some* (*getpid-from-inode-evevt e*)

definition *is-process-privileged* :: *'a*⇒*process-id* ⇒ *bool*
where *is-process-privileged s pid* ≡ *False*

definition *interference* :: *process-id* ⇒ *'a* ⇒*process-id* ⇒ *bool* ((- @ - -)) **where**
(*d1* @ *s d2*) ≡
(*if is-process-privileged s d1 then True*
else if d1 = *d2 then True*
else if (*alter s d1* ∩ *observe s d2*) ≠ {} *then True*
else False)

definition *kvpeq* :: *'a* ⇒ *process-id* ⇒ *'a* ⇒ *bool* ((- ∼ - ∼ -))
where *kvpeq s d t* ≡
((*subj-label s d*) = (*subj-label t d*)) ∧
(*observe s d*) = (*observe t d*) ∧
(∀ *v* .*interference v s d* = *interference v t d*)∧
(*let obs* = (*observe s d*) *in* ∀ *n*. *n* ∈*obs* ⟶ (*contents s n*) =
(*contents t n*))

definition *non-interference* :: *process-id* ⇒ *'a* ⇒ *process-id* ⇒ *bool* ((- @ - \ -))
where (*u* @ *s*\ *v*) ≡ ¬ (*u* @ *s v*)

declare *non-interference-def* [*cong*] **and** *domain-of-event-def* [*cong*]

lemma *kvpeq-transitive-lemma* : ∀ *s t r d*. (*kvpeq s d t*) ∧ (*kvpeq t d r*) ⟶ (*kvpeq s d r*)
by (*simp add*: *kvpeq-def*)

lemma *kvpeq-symmetric-lemma* : ∀ *s t d*. (*kvpeq s d t*) ⟶ (*kvpeq t d s*)
by (*simp add*: *kvpeq-def*)

lemma *kvpeq-reflexive-lemma* : ∀ *s d*. (*kvpeq s d s*)
by (*auto simp add*: *kvpeq-def*)

lemma *reachable-top*:
$\forall s\ a.\ (SM.reachable0\ s0\ exec\text{-}event\ )\ s \longrightarrow (\exists s'.\ \ s' = exec\text{-}event\ a)$
by *simp*

lemma *policy-respect1*: $\forall v\ d\ s\ t.\ (s \sim d \sim t)$
$\longrightarrow (interference\ v\ s\ d = interference\ v\ t\ d)$
using *kvpeq-def* **by** *auto*

definition *obsalter-cons* $\equiv \forall\ \ s\ t\ u\ v.\ (s \sim u \sim t) \wedge (\ s \sim v \sim t\ )$
$\longrightarrow (alter\ s\ v \cap observe\ s\ u) = (alter\ t\ v \cap observe\ t\ u)$

lemma *vpeq-def1*: $\forall\ \ s\ t\ u.\ (s \sim u \sim t) \longrightarrow$
$(\forall v\ .interference\ v\ s\ u = interference\ v\ t\ u) \wedge$
$(\forall\ \ n \in (observe\ s\ u).\ (contents\ s\ n) = (contents\ t\ n))$
by (*simp add*: *kvpeq-def*)

lemma *interf-reflexive-lemma* : $\forall d\ s.\ interference\ d\ s\ d$
using *interference-def* **by** *auto*

lemma *nintf-neq*: $u\ @\ s\backslash\ \ v \Longrightarrow u \neq v$
using *interf-reflexive-lemma non-interference-def* **by** *auto*

lemma *nintf-reflx*: *interference u s u*
by (*simp add*: *interf-reflexive-lemma*)

lemma *contents-consistent'* : $(\forall\ \ s\ u\ t.\ (s \sim u \sim t)\ \ \longrightarrow(\forall n \in observe\ s\ u.\ contents\ s\ n = contents\ t\ n))$
by (*simp add*: *vpeq-def1*)

lemma *observed-consistent'* : $(\forall\ \ s\ t\ u.\ ((s \sim u \sim t) \longrightarrow s\ \ \ u\ \ = t\ \ \ u))$
using *kvpeq-def* **by** *blast*

lemma *ac-interferes'* : $\forall\ \ s\ u\ v\ n.\ n \in alter\ s\ u \wedge n \in observe\ s\ v \longrightarrow (u\ @\ s\ \ v)$
using *interference-def* **by** *auto*

thm *LSM-Security-model.intro*
thm *SM-enabled.intro*
thm *LSM-Security-model-axioms.intro*
interpretation *LSM-Security-model s0 exec-event domain-of-event kvpeq interference observe alter contents*
using *kvpeq-transitive-lemma kvpeq-symmetric-lemma kvpeq-reflexive-lemma ac-interferes' nintf-reflx policy-respect1 reachable-top contents-consistent' observed-consistent' SM.intro*[*of kvpeq interference*]
SM-enabled-axioms.intro[*of s0 exec-event kvpeq interference*]
SM-enabled.intro[*of kvpeq interference*]
LSM-Security-model.intro[*of s0 exec-event kvpeq interference*]
LSM-Security-model-axioms.intro[*of kvpeq observe contents alter interference*]

by *fast*

29.22 Concrete unwinding condition of "local respect"

29.23 smack superblock hooks local respect proof

29.23.1 proving "sb$_{c}opy_{d}ata"satisfyingthe"localrespect"property$

lemma *k-sb-copy-data-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: s' = *fst* (*k-sb-copy-data s pid*)
shows $s \sim d \sim s'$
proof−
have *a1*: $s = s'$
apply (*simp add*: *p2 k-sb-copy-data-def*)
by (*metis* (*mono-tags*, *lifting*) *fstI*)
then show *?thesis*
by (*simp add*: *kvpeq-reflexive-lemma*)
qed

lemma *k-sb-copy-data-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SbEvt* (*Event-sb-copy-data pid sb*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: s' = *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
have *a1*: s' = *fst* (*k-sb-copy-data s pid*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 k-sb-copy-data-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *k-sb-copy-data-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-copy-data pid sb*))
using *dynamic-local-respect-e-def k-sb-copy-data-local-rsp-e non-interference-def*
by *blast*

29.23.2 proving "do$_r$emount" satisfyingthe" localrespect" property

lemma *do-remount-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: $\neg$(*interference pid s d*)
    and *p2*: $s' = fst$(*do-remount s p v*)
  shows $s \sim d \sim s'$
  proof−
    have *a1*: $s = s'$
      by (*simp add*: *p2 do-remount-def*)
    then show *?thesis*
      by (*simp add*: *kvpeq-reflexive-lemma*)
  qed

lemma *do-remount-local-rsp-e*:
   assumes *p0* :*reachable0 s*
    and *p1*: *e* = *SbEvt* (*Event-sb-remount pid p v*)
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: $s' =$ *exec-event s e*
  shows $s \sim d \sim s'$
    proof −
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
    have *a1*: $s' = fst$(*do-remount s p v*)
      using *p1 p3 exec-event-def* **by** *auto*
    have *a2*: $\neg$(*interference pid s d*)
      using *p2 a0 non-interference-def*
      by *blast*
    have *a3*: $s \sim d \sim s'$
      using *a1 a2 p0 do-remount-local-rsp* **by** *blast*
  }
  then show *?thesis*
    by *fast*
qed

lemma *do-remount-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-remount pid p v*))
  using *dynamic-local-respect-e-def do-remount-local-rsp-e non-interference-def* **by** *blast*

thm *mount-fs-def*

29.23.3 proving "mount$_f$s" satisfyingthe" localrespect" property

lemma *mount-fs-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: $\neg$(*interference pid s d*)
    and *p2*: $s' = fst$(*mount-fs s t f name data*)
  shows $s \sim d \sim s'$

proof−
  have *a1*: $s = s'$
    apply (*simp add*: *p2 mount-fs-def*)
    by (*smt fstI*)
  then show *?thesis*
    by (*simp add*: *kvpeq-reflexive-lemma*)
qed

lemma *mount-fs-local-rsp-e*:
  assumes *p0* :*reachable0 s*
  and *p1*: $e = SbEvt$ (*Event-sb-kern-mount pid t f name data*)
  and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
  and *p3*: $s' = exec\text{-}event\ s\ e$
  shows $s \sim d \sim s'$
  proof −
 {
  have *a0*: (*the* (*domain-of-event e*)) = *pid*
    using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
  have *a1*: $s' = fst$(*mount-fs s t f name data*)
    using *p1 p3 exec-event-def* **by** *auto*
  have *a2*: ¬(*interference pid s d*)
    using *p2 a0 non-interference-def*
    by *blast*
  have *a3*: $s \sim d \sim s'$
    using *a1 a2 p0 mount-fs-local-rsp* **by** *blast*
 }
 then show *?thesis*
  by *fast*
qed

lemma *mount-fs-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-kern-mount pid t f name data*))
  using *dynamic-local-respect-e-def mount-fs-local-rsp-e non-interference-def* **by** *blast*

29.23.4 proving "show$_sb_opts$"$satisfyingthe$"$localrespect$"$property$

lemma *k-show-sb-opts-local-rsp*:
 assumes *p0*: *reachable0 s*
  and *p1*: ¬(*interference pid s d*)
  and *p2*: $s' = fst$ (*show-sb-opts s pid sq t*)
 shows $s \sim d \sim s'$
 proof−
  have *a1*: $s = s'$
    by (*simp add*: *p2 show-sb-opts-def*)
  then show *?thesis*
    by (*simp add*: *kvpeq-reflexive-lemma*)
 qed

lemma *k-show-sb-opts-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SbEvt* (*Event-sb-show-options pid sq t*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = $ *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
have *a1*: $s' = $ *fst* (*show-sb-opts s pid sq t*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 k-show-sb-opts-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *k-show-sb-opts-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-show-options pid sq t*))
using *dynamic-local-respect-e-def k-show-sb-opts-local-rsp-e non-interference-def*
by *blast*

29.23.5 proving "sb$_{s}tatfs"satisfyingthe"localrespect"property$

lemma *k-sb-statfs-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = $ *fst* (*statfs-by-dentry s pid de*)
shows $s \sim d \sim s'$
proof–
have *a1*: $s = s'$
by (*simp add*: *p2 statfs-by-dentry-def*)
then show *?thesis*
by (*simp add*: *kvpeq-reflexive-lemma*)
qed

lemma *k-sb-statfs-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SbEvt* (*Event-sb-statfs pid de*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = $ *exec-event s e*
shows $s \sim d \sim s'$
proof –

{
  have *a0*: (*the* (*domain-of-event e*)) = *pid*
    using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
  have *a1*: *s′* = *fst* (*statfs-by-dentry s pid de*)
    using *p1 p3 exec-event-def* **by** *auto*
  have *a2*: ¬(*interference pid s d*)
    using *p2 a0 non-interference-def*
    by *blast*
  have *a3*: *s* ∼ *d* ∼ *s′*
    using *a1 a2 p0 k-sb-statfs-local-rsp* **by** *blast*
}
then show *?thesis*
  by *fast*
qed

lemma *sb-statfs-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-statfs pid de*))
  using *dynamic-local-respect-e-def k-sb-statfs-local-rsp-e non-interference-def* **by** *blast*

29.23.6 proving "do$_mount$"$satisfyingthe$"$localrespect$"$property$

lemma *do-mount-local-rsp*:
 assumes *p0*: *reachable0 s*
  and *p1*: ¬(*interference pid s d*)
  and *p2*: *s′* = *fst* (*do-mount s pid dev-name dir-name type-page flags′ data-page*)
 shows *s* ∼ *d* ∼ *s′*
 proof−
  have *a1*: *s* = *s′*
   by (*simp add*: *p2 do-mount-def*)
  then show *?thesis*
   by (*simp add*: *kvpeq-reflexive-lemma*)
 qed

lemma *do-mount-local-rsp-e*:
  assumes *p0* :*reachable0 s*
  and *p1*: *e* = *SbEvt* (*Event-sb-mount pid devname dirname t f p*)
  and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
  and *p3*: *s′* = *exec-event s e*
 shows *s* ∼ *d* ∼ *s′*
  proof −
 {
  have *a0*: (*the* (*domain-of-event e*)) = *pid*
    using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
  have *a1*: *s′* = *fst* (*do-mount s pid devname dirname t f p*)
    using *p1 p3 exec-event-def* **by** *auto*
  have *a2*: ¬(*interference pid s d*)
    using *p2 a0 non-interference-def*
    by *blast*

have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 do-mount-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *do-mount-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-mount pid devname dirname t f p*))
using *do-mount-local-rsp-e dynamic-local-respect-e-def non-interference-def* **by** *presburger*

29.23.7 proving "do$_u$*mount*"*satisfyingthe*"*localrespect*"*property*

lemma *do-umount-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst$ (*do-umount s pid m f*)
shows $s \sim d \sim s'$
proof−
have *a1*: $s = s'$
by (*simp add*: *p2 do-umount-def*)
then show *?thesis*
by (*simp add*: *kvpeq-reflexive-lemma*)
qed

lemma *do-umount-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *SbEvt* (*Event-sb-umount pid m f*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
have *a1*: $s' = fst$ (*do-umount s pid m f*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 do-umount-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *do-umount-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-umount*

pid m f))
  using *do-umount-local-rsp-e dynamic-local-respect-e-def non-interference-def*
  by *blast*

29.23.8 proving "pivot$_{r}oot" satisfying the "local respect" property$

lemma *pivot-root-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: $\neg$(*interference pid s d*)
    and *p2*: $s' = fst$ (*pivot-root s pid*)
  shows $s \sim d \sim s'$
  proof−
    have *a1*: $s = s'$
      by (*simp add*: *p2 pivot-root-def*)
    then show *?thesis*
      by (*simp add*: *kvpeq-reflexive-lemma*)
  qed

lemma *pivot-root-local-rsp-e*:
   assumes *p0* :*reachable0 s*
    and *p1*: *e* = *SbEvt* (*Event-sb-pivotroot pid*)
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: $s' = $ *exec-event s e*
  shows $s \sim d \sim s'$
    proof −
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
    have *a1*: $s' = fst$ (*pivot-root s pid*)
      using *p1 p3 exec-event-def* **by** *auto*
    have *a2*: $\neg$(*interference pid s d*)
      using *p2 a0 non-interference-def*
      by *blast*
    have *a3*: $s \sim d \sim s'$
      using *a1 a2 p0 pivot-root-local-rsp* **by** *blast*
  }
  then show *?thesis*
    by *fast*
qed

lemma *pivot-root-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-pivotroot pid*))
  using *pivot-root-local-rsp-e dynamic-local-respect-e-def non-interference-def*
  by *blast*

29.23.9 proving "setup$_{s}ecurity_{o}ptions" satisfying the "local respect" property$

lemma *setup-security-options-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: $\neg$(*interference pid s d*)

and *p2*: $s' = fst(setup\text{-}security\text{-}options\ s\ pid\ n\ sb\ opt)$
shows $s \sim d \sim s'$
using *p2 setup-security-options-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *setup-security-options-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SbEvt* (*Event-set-mnt-opts pid n sb opt*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
have *a1*: $s' = fst(setup\text{-}security\text{-}options\ s\ pid\ n\ sb\ opt)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 setup-security-options-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *setup-security-options-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-set-mnt-opts pid n sb opt*))
using *setup-security-options-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.23.10 proving "set$_sb_security$"$satisfyingthe$"$localrespect$"$property$

lemma *set-sb-security-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst\ (set\text{-}sb\text{-}security\ s\ pid\ sb\ de\ info)$
shows $s \sim d \sim s'$
proof−
have *a1*: $s = s'$
by (*simp add*: *p2 set-sb-security-def*)
then show *?thesis*
by (*simp add*: *kvpeq-reflexive-lemma*)
qed

lemma *set-sb-security-local-rsp-e*:
assumes *p0* :*reachable0 s*

and *p1*: *e* = *SbEvt* (*Event-set-sb-security pid sb de info*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-sb-event-def* **by** *auto*
have *a1*: *s'* = *fst* (*set-sb-security s pid sb de info*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 set-sb-security-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *set-sb-security-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-set-sb-security pid sb de info*))
using *set-sb-security-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.23.11 proving "$\textbf{nfs}_{c}lone_{s}b_{s}ecurity$" $satisfyingthe$" $localrespect$" $property$

lemma *nfs-clone-sb-security-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s'* = *fst*(*nfs-clone-sb-security s pid sb de minfo*)
shows $s \sim d \sim s'$
proof(*cases result s* (*security-sb-clone-mnt-opts' s oldsb sb' kflags kflags-out*))
case *True*
have *a1*: *s* = *s'*
using *p2 True* **apply**(*auto simp add*: *nfs-clone-sb-security-def*)
by (*smt fst-conv*)
then show *?thesis* **by** (*simp add*: *kvpeq-reflexive-lemma*)
next
case *False*
have *a1*: *s* = *s'* **using** *p2 False nfs-clone-sb-security-def*
by (*smt fst-conv*)
then show *?thesis* **by** (*simp add*: *kvpeq-reflexive-lemma*)
qed

lemma *nfs-clone-sb-security-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SbEvt* (*Event-sb-clone-mnt-opts pid sb de minfo*)

and *p2*:*non-interference* (*the*(*domain-of-event* *e*)) *s* *d*
and *p3*: $s' =$ *exec-event* *s* *e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event* *e*)) = *pid*
using *p1* *domain-of-event-def* *getpid-from-sb-event-def* **by** *auto*
have *a1*: $s' =$ *fst*(*nfs-clone-sb-security* *s* *pid* *sb* *de* *minfo*)
using *p1* *p3* *exec-event-def* **by** *auto*
have *a2*: ¬(*interference* *pid* *s* *d*)
using *p2* *a0* *non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1* *a2* *p0* *nfs-clone-sb-security-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *nfs-clone-sb-security-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-clone-mnt-opts* *pid* *sb* *d* *minfo*))
using *nfs-clone-sb-security-local-rsp-e* *dynamic-local-respect-e-def* *non-interference-def*

by *blast*

29.23.12 proving "parse$_security_options$" $satisfyingthe$" $localrespect$" $property$

lemma *parse-security-options-local-rsp*:
assumes *p0*: *reachable0* *s*
and *p1*: ¬(*interference* *pid* *s* *d*)
and *p2*: $s' =$ *fst*(*parse-security-options* *s* *pid* *str* *opts'*)
shows $s \sim d \sim s'$
proof−
have *a1*: $s = s'$
by (*simp* *add*: *p2* *parse-security-options-def*)
then show *?thesis*
by (*simp* *add*: *kvpeq-reflexive-lemma*)
qed

lemma *parse-security-options-local-rsp-e*:
assumes *p0* :*reachable0* *s*
and *p1*: *e* = *SbEvt* (*Event-sb-parse-opts-str* *pid* *str* *opts'*)
and *p2*:*non-interference* (*the*(*domain-of-event* *e*)) *s* *d*
and *p3*: $s' =$ *exec-event* *s* *e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event* *e*)) = *pid*
using *p1* *domain-of-event-def* *getpid-from-sb-event-def* **by** *auto*

have *a1*: $s' = fst(parse\text{-}security\text{-}options\ s\ \ pid\ str\ opts')$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg(interference\ pid\ s\ d)$
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 parse-security-options-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *parse-security-options-dlocal-rsp-e*: *dynamic-local-respect-e* (*SbEvt* (*Event-sb-parse-opts-str pid str opts'*))
using *parse-security-options-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.24 smack task hooks local respect proof

29.24.1 proving "prepare$_c$*reds*"*satisfyingthe*"*localrespect*"*property*

lemma *prepare-creds-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg(interference\ pid\ s\ d)$
and *p2*: $s' = fst(prepare\text{-}creds\ s\ pid)$
shows $s \sim d \sim s'$
using *p2 prepare-creds-def fst-conv vpeq-reflexive-lemma*
by *smt*

lemma *prepare-creds-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e = TskEvt\ (\ (Event\text{-}prepare\text{-}creds\ pid))$
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' =\ fst(prepare\text{-}creds\ s\ pid)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg(interference\ pid\ s\ d)$
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 prepare-creds-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*

qed

lemma *prepare-creds-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-prepare-creds pid*)))
  using *prepare-creds-local-rsp-e dynamic-local-respect-e-def non-interference-def*
  by *blast*

29.24.2 proving "sys$_{s}etreuid$" $satisfyingthe$" $localrespect$" $property$

lemma *sys-setreuid-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: $\neg$(*interference pid s d*)
    and *p2*: $s' = fst(sys\text{-}setreuid\ s\ pid\ kuid\ euid')$
  shows $s \sim d \sim s'$
proof(*cases snd* (*prepare-creds s pid*) = *None*)
  case *True*
  have *a1*: $s = s'$
    apply(*simp add*: *p2 sys-setreuid-def*)
    by (*simp add*: *True*)
  then show *?thesis* **by** (*simp add*: *vpeq-reflexive-lemma*)
next
  case *False*
  have *a1*: $s = s'$ **using** *p2 False*
    apply(*simp add*: *resultValue-def security-task-fix-setuid'-def the-run-state-def return-def*
      modify-def put-def get-def bind-def Let-def split-def LSM-SETID-RE-def ENOMEM-def
      prepare-creds-def)
    by (*smt fst-conv sys-setreuid-def*)
  then show *?thesis*
    using *kvpeq-reflexive-lemma* **by** *blast*
qed

lemma *sys-setreuid-local-rsp-e*:
  assumes *p0* :*reachable0 s*
    and *p1*: *e* = *TskEvt* ((*Event-sys-setreuid pid kuid euid'*))
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: $s' = exec\text{-}event\ s\ e$
  shows $s \sim d \sim s'$
    proof −
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
    have *a1*: $s' = fst(sys\text{-}setreuid\ s\ pid\ kuid\ euid')$
      using *p1 p3 exec-event-def*
      by *simp*
    have *a2*: $\neg$(*interference pid s d*)
      using *p2 a0 non-interference-def*

by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 sys-setreuid-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *sys-setreuid-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-sys-setreuid pid kuid euid'*)))
using *sys-setreuid-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.24.3 proving "setpgid" satisfying the "local respect" property

lemma *setpgid-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*setpgid s pid i pgid*)
shows $s \sim d \sim s'$
using *p2 setpgid-def fst-conv vpeq-reflexive-lemma*
by *smt*

lemma *setpgid-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *TskEvt* ((*Event-setpgid pid i pgid*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' = fst$(*setpgid s pid i pgid*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 setpgid-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *setpgid-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-setpgid pid i pgid*)))
using *setpgid-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

thm *do-getpgid-def*

29.24.4 proving "do$_getpgid$"$satisfyingthe$"$localrespect$"$property$

lemma *do-getpgid-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: ¬(*interference pid s d*)
    and *p2*: *s'* =*fst*(*do-getpgid s pid i*)
  shows *s* ∼ *d* ∼ *s'*
proof(*cases resultValue s* (*security-task-getpgid s p*) ≠ *0*)
  case *True*
  have *a1*: *s* = *s'*
      using *p2 True do-getpgid-def*
      by (*smt fst-conv*)
    then show *?thesis*
      by (*simp add*: *vpeq-reflexive-lemma*)
  next
  case *False*
  then show *?thesis*
   by (*smt do-getpgid-def p2 prod.simps*(*1*) *surjective-pairing vpeq-reflexive-lemma*)
qed

lemma *do-getpgid-local-rsp-e*:
   assumes *p0* :*reachable0 s*
    and *p1*: *e* = *TskEvt* ((*Event-do-getpgid pid i*))
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: *s'* = *exec-event s e*
  shows *s* ∼ *d* ∼ *s'*
    proof −
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
    have *a1*: *s'* = *fst*(*do-getpgid s pid i*)
      using *p1 p3 exec-event-def* **by** *auto*
    have *a2*: ¬(*interference pid s d*)
      using *p2 a0 non-interference-def*
      by *blast*
    have *a3*: *s* ∼ *d* ∼ *s'*
      using *a1 a2 p0 do-getpgid-local-rsp* **by** *blast*
  }
  then show *?thesis*
    by *fast*
qed

lemma *do-getpgid-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-do-getpgid pid i*)))
  using *do-getpgid-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.24.5 proving "getsid" satisfying the "local respect" property

lemma *getsid-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst(getsid\ s\ pid\ i)$
shows $s \sim d \sim s'$
using *fst-conv vpeq-reflexive-lemma p2 getsid-def*
by (*smt case-prod-conv*)

lemma *getsid-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *TskEvt* ((*Event-getsid pid i*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' = fst(getsid\ s\ pid\ i)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 getsid-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *getsid-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-getsid pid i*)))
using *getsid-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.24.6 proving "getsecid" satisfying the "local respect" property

lemma *getsecid-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst(getsecid\ s\ pid\ p\ u)$
shows $s \sim d \sim s'$
using *p2 getsecid-def* **by** (*simp add*: *vpeq-reflexive-lemma*)

lemma *getsecid-local-rsp-e*:
assumes *p0* :*reachable0 s*

and *p1*: *e* = *TskEvt* ((*Event-getsecid pid p u*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: *s'* = *fst*(*getsecid s pid p u*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 getsecid-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *getsecid-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-getsecid pid p u*)))
using *getsecid-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.24.7 proving "task$_{s}etnice$" $satisfying the$" $local respect$" $property$

lemma *task-setnice-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s'* = *fst*(*task-setnice s pid p i*)
shows $s \sim d \sim s'$
proof(*cases resultValue s* (*security-task-setnice s p nice*))
case (*nonneg n*)
have *a1*: *s* = *s'* **using** *p2 nonneg task-setnice-def*
by (*smt fst-conv*)
then show *?thesis* **by** (*simp add*: *vpeq-reflexive-lemma*)
next
case (*neg n*)
then show *?thesis* **using** *p2 neg task-setnice-def*
by (*smt fst-conv vpeq-reflexive-lemma*)
qed

lemma *task-setnice-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *TskEvt* ((*Event-task-setnice pid p i*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows $s \sim d \sim s'$

proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' = fst(task\text{-}setnice\ s\ pid\ p\ i)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 task-setnice-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *task-setnice-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-task-setnice pid p i*)))
using *task-setnice-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.24.8 proving "$\mathbf{set}_t ask_i oprio$"$satisfyingthe$"$localrespect$"$property$

lemma *set-task-ioprio-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst(set\text{-}task\text{-}ioprio\ s\ pid\ p\ i)$
shows $s \sim d \sim s'$
proof(*cases resultValue s* (*security-task-setioprio s p ioprio*))
case (*nonneg n*)
have *a1*: $s = s'$ **using** *p2 nonneg set-task-ioprio-def*
by (*smt fst-conv*)
then show *?thesis* **by** (*simp add*: *vpeq-reflexive-lemma*)
next
case (*neg n*)
then show *?thesis* **using** *p2 neg set-task-ioprio-def*
by (*smt fst-conv vpeq-reflexive-lemma*)
qed

lemma *set-task-ioprio-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *TskEvt* ((*Event-set-task-ioprio pid p i*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*

using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' = fst(set\text{-}task\text{-}ioprio\ s\ pid\ p\ i)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg(interference\ pid\ s\ d)$
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 set-task-ioprio-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *set-task-ioprio-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-set-task-ioprio pid p i*)))
using *set-task-ioprio-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.24.9 proving "get$_t$$ask_i$$oprio$"$satisfyingthe$"$localrespect$"$property$

lemma *get-task-ioprio-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg(interference\ pid\ s\ d)$
and *p2*: $s' = fst(get\text{-}task\text{-}ioprio\ s\ pid\ p)$
shows $s \sim d \sim s'$
using *fst-conv vpeq-reflexive-lemma p2 get-task-ioprio-def*
by *smt*

lemma *get-task-ioprio-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *TskEvt* ((*Event-get-task-ioprio pid p*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' = fst(get\text{-}task\text{-}ioprio\ s\ pid\ p\ )$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg(interference\ pid\ s\ d)$
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 get-task-ioprio-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*

qed

lemma *get-task-ioprio-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-get-task-ioprio pid p*)))
using *get-task-ioprio-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.24.10 proving "check$_p rlimit_p ermission$" $satisfying the$ "$local respect$" $property$

lemma *check-prlimit-permission-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*check-prlimit-permission s pid p i*)
shows $s \sim d \sim s'$
proof–
{
have *a1*: $s = s'$ **using** *p2 check-prlimit-permission-def*
by (*smt fst-conv*)
then show *?thesis* **by** (*simp add*: *vpeq-reflexive-lemma*)
}
qed

lemma *check-prlimit-permission-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *TskEvt* ((*Event-check-prlimit-permission pid p i*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' = fst$(*check-prlimit-permission s pid p i*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 check-prlimit-permission-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *check-prlimit-permission-dlocal-rsp-e*: *dynamic-local-respect-e*
(*TskEvt* ((*Event-check-prlimit-permission pid p i*)))
using *check-prlimit-permission-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.24.11 proving "do$_p rlimit$" $satisfyingthe$" $localrespect$" $property$

lemma *do-prlimit-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst(do\text{-}prlimit\ s\ pid\ p\ i)$
shows $s \sim d \sim s'$
using *p2 do-prlimit-def*
by (*simp add*: *vpeq-reflexive-lemma*)

lemma *do-prlimit-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *TskEvt* ((*Event-do-prlimit pid p i*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' = fst(do\text{-}prlimit\ s\ pid\ p\ i)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 do-prlimit-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *do-prlimit-dlocal-rsp-e*: *dynamic-local-respect-e*
(*TskEvt* ((*Event-do-prlimit pid p i*)))
using *do-prlimit-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.24.12 proving "task$_s etscheduler$" $satisfyingthe$" $localrespect$" $property$

lemma *task-setscheduler-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst(task\text{-}setscheduler\ s\ pid\ p)$
shows $s \sim d \sim s'$
using *p2 task-setscheduler-def*
by (*smt fstI vpeq-reflexive-lemma*)

lemma *task-setscheduler-local-rsp-e*:

assumes *p0* :*reachable0 s*
and *p1*: *e* =*TskEvt* ((*Event-task-setscheduler pid p*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' =$ *fst*(*task-setscheduler s pid p*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 task-setscheduler-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *task-setscheduler-dlocal-rsp-e*: *dynamic-local-respect-e*
(*TskEvt* ((*Event-task-setscheduler pid p*)))
using *task-setscheduler-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.24.13 proving "task$_getscheduler$" $satisfyingthe$" $localrespect$" $property$

lemma *task-getscheduler-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' =$ *fst*(*task-getscheduler s pid p*)
shows $s \sim d \sim s'$
using *p2 task-getscheduler-def*
by (*smt fstI vpeq-reflexive-lemma*)

lemma *task-getscheduler-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *TskEvt* ((*Event-task-getscheduler pid p*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' =$ *fst*(*task-getscheduler s pid p*)
using *p1 p3 exec-event-def* **by** *auto*

have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 task-getscheduler-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *task-getscheduler-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-task-getscheduler pid p*)))
using *dynamic-local-respect-e-def non-interference-def task-getscheduler-local-rsp-e*
by *blast*

29.24.14 proving "task$_movememory$" $satisfyingthe$" $localrespect$" $property$

lemma *task-movememory-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*task-movememory s pid p*)
shows $s \sim d \sim s'$
using *p2 task-movememory-def*
by (*smt fstI vpeq-reflexive-lemma*)

lemma *task-movememory-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *TskEvt* ((*Event-task-movememory pid p*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' = fst$(*task-movememory s pid p*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 task-movememory-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *task-movememory-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-task-movememory pid p*)))

using *task-movememory-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.24.15 proving "task$_kill$" $satisfyingthe$" $localrespect$" $property$

lemma *task-kill-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst(task\text{-}kill\ s\ pid\ p\ sinfo\ i\ c)$
shows $s \sim d \sim s'$
using *p2 task-kill-def*
by (*smt fstI vpeq-reflexive-lemma*)

lemma *task-kill-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e = TskEvt\ ((Event\text{-}task\text{-}kill\ pid\ p\ sinfo\ i\ c))$
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: $s' = fst(task\text{-}kill\ s\ pid\ p\ sinfo\ i\ c)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 task-kill-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *task-kill-dlocal-rsp-e*: *dynamic-local-respect-e* (*TskEvt* ((*Event-task-kill pid p sinfo i c*)))
using *task-kill-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *presburger*

29.24.16 proving "task$_prctl$" $satisfyingthe$" $localrespect$" $property$

lemma *task-prctl-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst(task\text{-}prctl\ s\ pid\ op\ arg2\ arg3\ arg4\ arg5)$
shows $s \sim d \sim s'$
using *p2 task-prctl-def*
by (*smt fst-conv vpeq-reflexive-lemma*)

lemma *task-prctl-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *TskEvt* ((*Event-task-prctl pid op arg2 arg3 arg4 arg5*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* ∼ *d* ∼ *s'*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-tsk-event-def* **by** *auto*
have *a1*: *s'* = *fst*(*task-prctl s pid op arg2 arg3 arg4 arg5*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s'*
using *a1 a2 p0 task-prctl-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *task-prctl-dlocal-rsp-e*: *dynamic-local-respect-e*
(*TskEvt* ((*Event-task-prctl pid op arg2 arg3 arg4 arg5*)))
using *task-prctl-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *presburger*

29.25 smack prtace hooks local respect proof

29.25.1 proving "ptrace$_{m}ay_{a}ccess$"$satisfyingthe$"$localrespect$"$property$

lemma *ptrace-may-access-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s'* = *fst*(*ptrace-may-access s pid p m*)
shows *s* ∼ *d* ∼ *s'*
proof−
have *a1*: *s* = *s'*
by (*simp add*: *p2 ptrace-may-access-def*)
then show *?thesis*
using *vpeq-reflexive-lemma* **by** *auto*
qed

lemma *ptrace-may-access-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *PtraceEvt* (*Event-ptrace-access-check pid p m*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* ∼ *d* ∼ *s'*

proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-ptrace-event-def* **by** *auto*
have *a1*: s' = *fst*(*ptrace-may-access s pid p m*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 ptrace-may-access-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *ptrace-may-access-dlocal-rsp-e*: *dynamic-local-respect-e* (*PtraceEvt* (*Event-ptrace-access-check pid p m*))
using *dynamic-local-respect-e-def ptrace-may-access-local-rsp-e non-interference-def*
by *blast*

29.25.2 proving "ptrace$_{t}raceme$"$satisfyingthe$"$localrespect$"$property$

lemma *ptrace-traceme-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: s' = *fst*(*ptrace-traceme s pid*)
shows $s \sim d \sim s'$
proof–
have *a1*: $s = s'$
by (*simp add*: *p2 ptrace-traceme-def*)
then show *?thesis*
by (*simp add*: *kvpeq-reflexive-lemma*)
qed

lemma *ptrace-traceme-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *PtraceEvt* (*Event-ptrace-traceme pid*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: s' = *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-ptrace-event-def* **by** *auto*
have *a1*: s' = *fst*(*ptrace-traceme s pid*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*

by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 ptrace-traceme-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *ptrace-traceme-dlocal-rsp-e*: *dynamic-local-respect-e* (*PtraceEvt* (*Event-ptrace-traceme pid*))
using *dynamic-local-respect-e-def ptrace-traceme-local-rsp-e non-interference-def*
by *blast*

29.25.3 proving "prepare$_binprm$"$satisfyingthe$"$localrespect$"$property$

lemma *prepare-binprm-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*prepare-binprm s pid bprm*)
shows $s \sim d \sim s'$
using *p2 prepare-binprm-def*
by (*smt fst-conv vpeq-reflexive-lemma*)

lemma *prepare-binprm-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *SysEvt*(*Event-prepare-binprm pid bprm*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-sys-event-def* **by** *auto*
have *a1*: $s' = fst$(*prepare-binprm s pid bprm*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 prepare-binprm-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *prepare-binprm-dlocal-rsp-e*: *dynamic-local-respect-e* (*SysEvt*(*Event-prepare-binprm pid bprm*))
using *prepare-binprm-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.26 smack file hooks local respect proof

lemma *do-ioctl-detstate*:
$\bigwedge sa$. $\{\!|\lambda s$. $s = sa|\!\}$ *security-file-ioctl sa file cmd arg* $\{\!|\lambda r\ s.\ s = sa|\!\} \longrightarrow$
snd (*the-run-state* (*security-file-ioctl sa file cmd arg*) *sa*)= *sa*

apply(*simp add*: *valid-def*)
apply (*simp add*: *security-file-ioctl-def the-run-state-def split-def*)
apply *auto*[*1*]
done

29.26.1 proving "do$_i$*octl*"*satisfyingthe*"*localrespect*"*property*

lemma *do-ioctl-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*do-ioctl s pid file cmd arg*)
shows $s \sim d \sim s'$
proof−
have *a1*: $s = s'$
apply (*simp add*: *p2 do-ioctl-def*)
using *do-ioctl-detstate fst-conv funcState-def security-file-ioctl-notchgstate*
by *smt*
then show *?thesis*
using *vpeq-reflexive-lemma* **by** *auto*
qed

lemma *do-ioctl-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *FileEvt*(*Event-do-ioctl pid file cmd arg*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: $s' = fst$(*do-ioctl s pid file cmd arg*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a4*: $s \sim d \sim s'$
using *a1 a2 p0 do-ioctl-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*

qed

lemma *do-ioctl-dlocal-rsp-e*: *dynamic-local-respect-e* (*FileEvt*(*Event-do-ioctl* *pid* *file cmd arg*))
  using *do-ioctl-local-rsp-e dynamic-local-respect-e-def non-interference-def*
  by *blast*

29.26.2 proving "syscall$_ioctl" satisfying the" local respect" property$

lemma *syscall-ioctl-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: $\neg$(*interference pid s d*)
    and *p2*: $s' = fst$(*syscall-ioctl s pid fd cmd arg*)
  shows $s \sim d \sim s'$
  using *p2 syscall-ioctl-def security-file-ioctl-def*
  by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *syscall-ioctl-local-rsp-e*:
   assumes *p0* :*reachable0 s*
    and *p1*: $e =$ *FileEvt*(*Event-syscall-ioctl pid fd cmd arg*)
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: $s' =$ *exec-event s e*
  shows $s \sim d \sim s'$
    proof –
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
    have *a1*: $s' = fst$(*syscall-ioctl s pid fd cmd arg*)
      using *p1 p3 exec-event-def* **by** *auto*
    have *a2*: $\neg$(*interference pid s d*)
      using *p2 a0 non-interference-def*
      by *blast*
    have *a3*: $s \sim d \sim s'$
      using *a1 a2 p0 syscall-ioctl-local-rsp* **by** *blast*
  }
  then show *?thesis*
    by *fast*
qed

lemma *syscall-ioctl-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-syscall-ioctl* *pid fd cmd arg*))
  using *syscall-ioctl-local-rsp-e dynamic-local-respect-e-def non-interference-def*
  by *blast*

29.26.3 proving "ksys$_ioctl" satisfying the" local respect" property$

lemma *ksys-ioctl-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: $\neg$(*interference pid s d*)
    and *p2*: $s' = fst$(*ksys-ioctl s pid fd cmd arg*)

shows $s \sim d \sim s'$
using *p2 ksys-ioctl-def security-file-ioctl-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *ksys-ioctl-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: e = *FileEvt*(*Event-ksys-ioctl pid fd cmd arg*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: s' = *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: s' = *fst*(*ksys-ioctl s pid fd cmd arg*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 ksys-ioctl-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *ksys-ioctl-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-ksys-ioctl pid fd cmd arg*))
using *ksys-ioctl-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.26.4 proving "vm$_m map_p goff$"$satisfyingthe$"$localrespect$"$property$

lemma *vm-mmap-pgoff-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: s' = *fst*(*vm-mmap-pgoff s pid file addr len' prot flag pgoff*)
shows $s \sim d \sim s'$
using *p2 vm-mmap-pgoff-def security-mmap-file-def fst-conv kvpeq-reflexive-lemma*
by *metis*

lemma *vm-mmap-pgoff-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: e = *FileEvt*(*Event-vm-mmap-pgoff pid file addr len' prot flag pgoff*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: s' = *exec-event s e*
shows $s \sim d \sim s'$
proof −
{

have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: *s'* = *fst*(*vm-mmap-pgoff s pid file addr len' prot flag pgoff*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s'*
using *a1 a2 p0 vm-mmap-pgoff-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *vm-mmap-pgoff-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-vm-mmap-pgoff pid file addr len' prot flag pgoff*))
using *vm-mmap-pgoff-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *presburger*

29.26.5 proving "do$_s ys_v m86$" $satisfyingthe$" $localrespect$" $property$

lemma *do-sys-vm86-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s'* = *fst*(*do-sys-vm86 s pid*)
shows *s* ∼ *d* ∼ *s'*
using *p2 do-sys-vm86-def security-mmap-addr-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *do-sys-vm86-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *FileEvt*(*Event-do-sys-vm86 pid*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* ∼ *d* ∼ *s'*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: *s'* = *fst*(*do-sys-vm86 s pid*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s'*
using *a1 a2 p0 do-sys-vm86-local-rsp* **by** *blast*
}
then show *?thesis*

by *fast*
qed

lemma *do-sys-vm86-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-do-sys-vm86* *pid*))
using *do-sys-vm86-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.26.6 proving "get$_u$nmapped$_a$rea" satisfying the "local respect" property

lemma *get-unmapped-area-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*get-unmapped-area s pid file addr*)
shows $s \sim d \sim s'$
using *p2 get-unmapped-area-def security-mmap-addr-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *get-unmapped-area-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *FileEvt*(*Event-get-unmapped-area pid file addr*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: $s' = fst$(*get-unmapped-area s pid file addr*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 get-unmapped-area-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *get-unmapped-area-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-get-unmapped-area* *pid file addr*))
using *get-unmapped-area-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.26.7 proving "validate$_m$map$_r$equest" satisfying the "local respect" property

lemma *validate-mmap-request-local-rsp*:
assumes *p0*: *reachable0 s*

and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst$(*validate-mmap-request s pid file addr*)
shows $s \sim d \sim s'$
using *p2 validate-mmap-request-def security-mmap-addr-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *validate-mmap-request-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *FileEvt*(*Event-validate-mmap-request pid file addr*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: $s' = fst$(*validate-mmap-request s pid file addr*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 validate-mmap-request-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *validate-mmap-request-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-validate-mmap-request pid file addr*))
using *validate-mmap-request-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.26.8 proving "generic$_s$etlease"$satisfyingthe$"$localrespect$"$property$

lemma *generic-setlease-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst$(*generic-setlease s pid file arg*)
shows $s \sim d \sim s'$
using *p2 generic-setlease-def security-file-lock-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *generic-setlease-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *FileEvt*(*Event-generic-setlease pid file arg*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*

shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: *s'* = *fst*(*generic-setlease s pid file arg*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 generic-setlease-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *generic-setlease-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-generic-setlease pid file arg*))
using *generic-setlease-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.26.9 proving "syscall$_{l}$*ock*"*satisfyingthe*"*localrespect*"*property*

lemma *syscall-lock-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s'* = *fst*(*syscall-lock s pid fd cmd*)
shows $s \sim d \sim s'$
using *p2 syscall-lock-def security-file-lock-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *syscall-lock-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *FileEvt*(*Event-syscall-lock pid fd cmd*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: *s'* = *fst*(*syscall-lock s pid fd cmd*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$

using *a1 a2 p0 syscall-lock-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *syscall-lock-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-syscall-lock pid fd cmd*))
using *syscall-lock-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.26.10 proving "$\mathbf{do}_lock_file_wait$"satisfyingthe"localrespect"property

lemma *do-lock-file-wait-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst(do\text{-}lock\text{-}file\text{-}wait\ s\ pid\ file\ cmd\ fl)$
shows $s \sim d \sim s'$
using *p2 do-lock-file-wait-def security-file-lock-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *do-lock-file-wait-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *FileEvt*(*Event-do-lock-file-wait pid file cmd fl*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: $s' = fst(do\text{-}lock\text{-}file\text{-}wait\ s\ pid\ file\ cmd\ fl)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a4*: $s \sim d \sim s'$
using *a1 a2 p0 do-lock-file-wait-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *do-lock-file-wait-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-do-lock-file-wait pid file cmd fl*))
using *do-lock-file-wait-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.26.11 proving "file$_f cntl$" $satisfying the$" $local respect$" $property$

lemma *file-fcntl-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: ¬(*interference pid s d*)
    and *p2*: $s' = fst(\textit{file-fcntl s pid file cmd arg})$
  shows $s \sim d \sim s'$
  using *p2 file-fcntl-def security-file-lock-def*
  by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *file-fcntl-local-rsp-e*:
  assumes *p0* :*reachable0 s*
    and *p1*: *e* = *FileEvt*(*Event-file-fcntl pid file cmd arg*)
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: $s' = \textit{exec-event } s\ e$
  shows $s \sim d \sim s'$
    proof −
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
    have *a1*: $s' = fst(\textit{file-fcntl s pid file cmd arg})$
      using *p1 p3 exec-event-def* **by** *auto*
    have *a2*: ¬(*interference pid s d*)
      using *p2 a0 non-interference-def*
      by *blast*
    have *a3*: $s \sim d \sim s'$
      using *a1 a2 p0 file-fcntl-local-rsp* **by** *blast*
  }
  then show *?thesis*
    by *fast*
qed

lemma *file-fcntl-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-file-fcntl pid file cmd arg*))
  using *file-fcntl-local-rsp-e dynamic-local-respect-e-def non-interference-def*
  by *blast*

29.26.12 proving "file$_s end_s igiotask$" $satisfying the$" $local respect$" $property$

lemma *file-send-sigiotask-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: ¬(*interference pid s d*)
    and *p2*: $s' = fst(\textit{file-send-sigiotask s pid t fown sig})$
  shows $s \sim d \sim s'$
  using *p2 file-send-sigiotask-def security-file-send-sigiotask-def*
  by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *file-send-sigiotask-local-rsp-e*:
  assumes *p0* :*reachable0 s*
    and *p1*: *e* = *FileEvt*(*Event-file-send-sigiotask pid t fown sig*)

and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: $s' =$ *fst*(*file-send-sigiotask s pid t fown sig*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 file-send-sigiotask-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *file-send-sigiotask-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-file-send-sigiotask pid t fown sig*))
using *file-send-sigiotask-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.26.13 proving "file$_receive$" satisfying the "local respect" property

lemma *file-receive-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' =$ *fst*(*file-receive s pid f*)
shows $s \sim d \sim s'$
using *p2 file-receive-def security-file-receive-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *file-receive-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *FileEvt* (*Event-file-receive pid f*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: $s' =$ *fst*(*file-receive s pid f*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*

by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 file-receive-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *file-receive-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt* (*Event-file-receive pid f*))
using *file-receive-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.26.14 proving "do$_dentry_open$"$satisfyingthe$"$localrespect$"$property$

lemma *do-dentry-open-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*do-dentry-open s pid f*)
shows $s \sim d \sim s'$
using *p2 do-dentry-open-def security-file-open-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *do-dentry-open-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *FileEvt*(*Event-do-dentry-open pid f*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = $ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-file-event-def* **by** *auto*
have *a1*: $s' = fst$(*do-dentry-open s pid f*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 do-dentry-open-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *do-dentry-open-dlocal-rsp-e*: *dynamic-local-respect-e*(*FileEvt*(*Event-do-dentry-open pid f*))
using *do-dentry-open-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.27 smack ipc hooks local respect proof

29.27.1 proving "ipcperms" satisfying the "local respect" property

lemma *ipcperms-local-rsp*:
$\llbracket$*reachable0 s*;¬(*interference pid s d*);*s'* = *fst*(*ipcperms s pid ipcp flg*)$\rrbracket$
$\Longrightarrow$(*s* ∼ *d* ∼ *s'*)
using *ipcperms-def security-ipc-permission-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *ipcperms-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *KIpcEvt*((*Event-ipc-permission pid ipcp flg*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* ∼ *d* ∼ *s'*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: *s'* = *fst*(*ipcperms s pid ipcp flg*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s'*
using *a1 a2 p0 ipcperms-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *ipcperms-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-ipc-permission pid ipcp flg*)))
using *ipcperms-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.27.2 proving "audit$_{ipc_obj}$"$satisfyingthe$"$localrespect$"$property$

lemma *audit-ipc-obj-local-rsp*:
$\llbracket$*reachable0 s*;¬(*interference pid s d*);*s'* = *fst*(*audit-ipc-obj s pid ipcp*)$\rrbracket$ $\Longrightarrow$(*s* ∼ *d* ∼ *s'*)
using *audit-ipc-obj-def security-ipc-getsecid-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *audit-ipc-obj-local-rsp-e*:
assumes *p0* :*reachable0 s*

and *p1*: $e = KIpcEvt((Event\text{-}ipc\text{-}getsecid\ pid\ ipcp)\ )$
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: $s' = fst(audit\text{-}ipc\text{-}obj\ s\ pid\ ipcp)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 audit-ipc-obj-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *audit-ipc-obj-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-ipc-getsecid pid ipcp*)))
using *audit-ipc-obj-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.27.3 proving "ksys$_{m}sgget$" $satisfyingthe$" $localrespect$" $property$

lemma *ksys-msgget-local-rsp*: ⟦*reachable0 s*; ¬(*interference pid s d*); $s' = fst(ksys\text{-}msgget\ s\ pid\ msq\ msqflg)$⟧
$\Longrightarrow (s \sim d \sim s')$
using *ksys-msgget-def security-msg-queue-associate-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *ksys-msgget-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e = KIpcEvt(\ (Event\text{-}msg\text{-}queue\text{-}associate\ pid\ msq\ msqflg\ ))$
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: $s' = fst(ksys\text{-}msgget\ s\ pid\ msq\ msqflg)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$

using *a1 a2 p0 ksys-msgget-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *ksys-msgget-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-msg-queue-associate pid msq msqflg*)))
using *ksys-msgget-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.27.4 proving "msg$_q$ueue$_m$sgctl" satisfyingthe" localrespect" property

lemma *msg-queue-msgctl-local-rsp*:
⟦*reachable0 s*;¬(*interference pid s d*);*s′* = *fst*(*msg-queue-msgctl s pid msq cmd*)⟧
⟹(*s* ∼ *d* ∼ *s′*)
using *msg-queue-msgctl-def security-msg-queue-msgctl-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *msg-queue-msgctl-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *KIpcEvt*((*Event-msg-queue-msgctl pid msq cmd*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s′* = *exec-event s e*
shows *s* ∼ *d* ∼ *s′*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: *s′* = *fst*(*msg-queue-msgctl s pid msq cmd*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s′*
using *a1 a2 p0 msg-queue-msgctl-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *msg-queue-msgctl-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-msg-queue-msgctl pid msq cmd*)))
using *msg-queue-msgctl-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.27.5 proving "do$_m$sgsnd" satisfyingthe" localrespect" property

lemma *do-msgsnd-local-rsp*:

assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*do-msgsnd s pid msq msg msgflg*)
shows $s \sim d \sim s'$
using *p2 do-msgsnd-def security-msg-queue-msgsnd-def*
proof −
have $\forall$ *s n k m i*. *do-msgsnd s n k m i* =
(*if resultValue s* (*security-msg-queue-msgsnd s k m i*) = *0*
then (*s*, *0*)
else
(*s*, *resultValue s* (*security-msg-queue-msgsnd s k m i*)))
by (*simp add*: *do-msgsnd-def*)
then have *do-msgsnd s pid msq msg msgflg* =
(*s*, *resultValue s* (*security-msg-queue-msgsnd s msq msg msgflg*))
by *presburger*
then show *?thesis*
using *p2*
by (*simp add*: *kvpeq-reflexive-lemma*)
qed

lemma *do-msgsnd-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *KIpcEvt*((*Event-msg-queue-msgsnd pid msq msg msgflg*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: s' = *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: $s' = fst$(*do-msgsnd s pid msq msg msgflg*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 do-msgsnd-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *do-msgsnd-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-msg-queue-msgsnd pid msq msg msgflg*)))
using *do-msgsnd-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.27.6 proving "msg$_q$$ueue_msgrcv$" $satisfyingthe$" $localrespect$" $property$

lemma *msg-queue-msgrcv-local-rsp*:
⟦*reachable0 s*; ¬(*interference pid s d*); *s′* = *fst*(*msg-queue-msgrcv s pid isp msq p long msqflg*)⟧
⟹(*s* ∼ *d* ∼ *s′*)
using *msg-queue-msgrcv-def security-msg-queue-msgrcv-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *msg-queue-msgrcv-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *KIpcEvt*((*Event-msg-queue-msgrcv pid isp msq p long msqflg*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s′* = *exec-event s e*
shows *s* ∼ *d* ∼ *s′*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: *s′* = *fst*(*msg-queue-msgrcv s pid isp msq p long msqflg*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s′*
using *a1 a2 p0 msg-queue-msgrcv-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *msg-queue-msgrcv-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-msg-queue-msgrcv pid isp msq p long msqflg*)))
using *msg-queue-msgrcv-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *presburger*

29.27.7 proving "ksys$_s$$hmget$" $satisfyingthe$" $localrespect$" $property$

lemma *ksys-shmget-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s′* = *fst*(*ksys-shmget s pid shm shmflg*)
shows *s* ∼ *d* ∼ *s′*
using *p2 ksys-shmget-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *ksys-shmget-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *KIpcEvt*((*Event-shm-associate pid shm shmflg*))

and *p2*:*non-interference* (*the*(*domain-of-event* *e*)) *s* *d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event* *e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: $s' = fst(ksys\text{-}shmget\ s\ pid\ shm\ shmflg)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 ksys-shmget-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *ksys-shmget-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-shm-associate pid shm shmflg*)))
using *ksys-shmget-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.27.8 proving "sem$_m$*sgctl*"*satisfyingthe*"*localrespect*"*property*

lemma *sem-msgctl-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst(sem\text{-}msgctl\ s\ pid\ sem\ cmd)$
shows $s \sim d \sim s'$
using *p2 sem-msgctl-def security-sem-semctl-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *sem-msgctl-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *KIpcEvt*((*Event-sem-semctl pid sem cmd*))
and *p2*:*non-interference* (*the*(*domain-of-event* *e*)) *s* *d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event* *e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: $s' = fst(sem\text{-}msgctl\ s\ pid\ sem\ cmd)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*

have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 sem-msgctl-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *sem-msgctl-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-sem-semctl pid sem cmd*)))
using *sem-msgctl-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.27.9 proving "do$_{s}$*emtimedop*" *satisfyingthe*" *localrespect*" *property*

lemma *do-semtimedop-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*do-semtimedop s pid sma sops nsops alter'*)
shows $s \sim d \sim s'$
using *p2 do-semtimedop-def*
by (*smt fstI kvpeq-reflexive-lemma*)

lemma *do-semtimedop-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *KIpcEvt*((*Event-sem-semop pid sma sops nsops alter'*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: $s' = fst$(*do-semtimedop s pid sma sops nsops alter'*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 do-semtimedop-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *do-semtimedop-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-sem-semop pid sma sops nsops alter'*)))
using *do-semtimedop-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *metis*

29.27.10 proving "do$_s$*hmat*" *satisfyingthe*" *localrespect*" *property*

lemma *do-shmat-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: *s'* = *fst*(*do-shmat s pid shp shmaddr shmflg*)
shows *s* $\sim$ *d* $\sim$ *s'*
proof−
have *a1*: *s* = *s'*
apply (*simp add*: *p2 do-shmat-def*)
by (*smt fst-conv*)
then show *?thesis*
by (*simp add*: *kvpeq-reflexive-lemma*)
qed

lemma *do-shmat-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *KIpcEvt*((*Event-shm-shmat pid shp shmaddr shmflg*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* $\sim$ *d* $\sim$ *s'*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: *s'* = *fst*(*do-shmat s pid shp shmaddr shmflg*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* $\sim$ *d* $\sim$ *s'*
using *a1 a2 p0 do-shmat-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *do-shmat-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-shm-shmat pid shp shmaddr shmflg*)))
using *do-shmat-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.27.11 proving "ksys$_s$*emget*" *satisfyingthe*" *localrespect*" *property*

lemma *ksys-semget-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: *s'* = *fst*(*ksys-semget s pid sem semflg*)
shows *s* $\sim$ *d* $\sim$ *s'*
using *p2 ksys-semget-def*

by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *ksys-semget-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e = KIpcEvt($ (*Event-sem-associate pid sem semflg*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: $s' = fst(ksys\text{-}semget\ s\ pid\ sem\ semflg)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 ksys-semget-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *ksys-semget-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-sem-associate pid sem semflg*)))
using *ksys-semget-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.27.12 proving "shm$_m$sgctl"$satisfyingthe$"$localrespect$"$property$

lemma *shm-msgctl-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' = fst(shm\text{-}msgctl\ s\ pid\ shm\ cmd)$
shows $s \sim d \sim s'$
using *p2 shm-msgctl-def security-shm-shmctl-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *shm-msgctl-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e = KIpcEvt($ (*Event-shm-shmctl pid shm cmd*))
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*

using *p1 domain-of-event-def getpid-from-kern-ipc-event-def* **by** *auto*
have *a1*: $s' = fst(shm\text{-}msgctl\ s\ pid\ shm\ cmd)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg(interference\ pid\ s\ d)$
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 shm-msgctl-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *shm-msgctl-dlocal-rsp-e*: *dynamic-local-respect-e*(*KIpcEvt*((*Event-shm-shmctl pid shm cmd*)))
using *shm-msgctl-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.28 smack key hooks local respect proof

29.28.1 proving "key$_{t}ask_{p}ermission$"$satisfyingthe$"$localrespect$"$property$

lemma *key-task-permission-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg(interference\ pid\ s\ d)$
and *p2*: $s' = fst(key\text{-}task\text{-}permission\ s\ pid\ key\text{-}ref\ cred'\ perm)$
shows $s \sim d \sim s'$
using *p2 key-task-permission-def security-key-permission-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *key-task-permission-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e = KeyEvt($ *Event-key-permission pid key-ref cred′ perm*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-key-evevt-def* **by** *auto*
have *a1*: $s' = fst(key\text{-}task\text{-}permission\ s\ pid\ key\text{-}ref\ cred'\ perm)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg(interference\ pid\ s\ d)$
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 key-task-permission-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*

qed

lemma *key-task-permission-dlocal-rsp-e*: *dynamic-local-respect-e*(*KeyEvt*(*Event-key-permission pid key-ref cred′ perm*))
using *key-task-permission-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.28.2 proving "keyctl$_{g}et_{s}ecurity$" $satisfyingthe$" $localrespect$" $property$

lemma *keyctl-get-security-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s′* = *fst*(*keyctl-get-security s pid keyid′ buffer buflen*)
shows *s* ∼ *d* ∼ *s′*
using *p2 keyctl-get-security-def security-key-getsecurity-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *keyctl-get-security-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *KeyEvt*(*Event-key-getsecurity pid keyid′ buffer buflen*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s′* = *exec-event s e*
shows *s* ∼ *d* ∼ *s′*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-key-evevt-def* **by** *auto*
have *a1*: *s′* = *fst*(*keyctl-get-security s pid keyid′ buffer buflen*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s′*
using *a1 a2 p0 keyctl-get-security-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *keyctl-get-security-dlocal-rsp-e*: *dynamic-local-respect-e*(*KeyEvt*(*Event-key-getsecurity pid keyid′ buffer buflen*))
using *keyctl-get-security-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.28.3 proving "check$_{s}yslog_{p}ermissions$" $satisfyingthe$" $localrespect$" $property$

lemma *check-syslog-permissions-local-rsp*:
assumes *p0*: *reachable0 s*

and *p1*: ¬(*interference pid s d*)
and *p2*: *s' = fst*(*check-syslog-permissions s pid t*)
shows *s* ∼ *d* ∼ *s'*
using *p2 check-syslog-permissions-def*
by (*smt fst-conv vpeq-reflexive-lemma*)

lemma *check-syslog-permissions-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SysEvt*(*Event-smack-syslog pid t*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* ∼ *d* ∼ *s'*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-sys-event-def* **by** *auto*
have *a1*: *s'* = *fst*(*check-syslog-permissions s pid t*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s'*
using *a1 a2 p0 check-syslog-permissions-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *check-syslog-permissions-dlocal-rsp-e*: *dynamic-local-respect-e* (*SysEvt*(*Event-smack-syslog pid t*))
using *check-syslog-permissions-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.29 smack aduit hooks local respect proof

29.29.1 proving "audit$_{d}ata_{t}o_{e}ntry$" $satisfyingthe$" $localrespect$" $property$

lemma *audit-data-to-entry-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s'* = *fst*(*audit-data-to-entry s pid*)
shows *s* ∼ *d* ∼ *s'*
using *p2 audit-data-to-entry-def security-audit-rule-init-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *audit-data-to-entry-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *AuditEvt*(*Event-audit-data-to-entry pid*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*

and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-aduit-evevt-def* **by** *auto*
have *a1*: $s' =$ *fst*(*audit-data-to-entry s pid*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 audit-data-to-entry-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *audit-data-to-entry-dlocal-rsp-e*: *dynamic-local-respect-e*(*AuditEvt*(*Event-audit-data-to-entry pid*))
using *audit-data-to-entry-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.29.2 proving "audit$_dupe_lsm_field$" $satisfying the$ " $local respect$ " $property$

lemma *audit-dupe-lsm-field-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' =$ *fst*(*audit-dupe-lsm-field s pid df sf*)
shows $s \sim d \sim s'$
proof–
have *a1*: $s = s'$
apply (*simp add*: *p2 audit-dupe-lsm-field-def*)
by (*smt eq-fst-iff*)
then show *?thesis*
using *vpeq-reflexive-lemma* **by** *auto*
qed

lemma *audit-dupe-lsm-field-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *AuditEvt*(*Event-audit-dupe-lsm-field pid df sf*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-aduit-evevt-def* **by** *auto*

have *a1*: $s' = fst(audit\text{-}dupe\text{-}lsm\text{-}field\ s\ pid\ df\ sf)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg(interference\ pid\ s\ d)$
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 audit-dupe-lsm-field-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *audit-dupe-lsm-field-dlocal-rsp-e*: *dynamic-local-respect-e*(*AuditEvt*(*Event-audit-dupe-lsm-field pid df sf*))
using *audit-dupe-lsm-field-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.29.3 proving "update$_lsm_rule$" satisfyingthe" localrespect" property

lemma *update-lsm-rule-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg(interference\ pid\ s\ d)$
and *p2*: $s' = fst(update\text{-}lsm\text{-}rule\ s\ pid\ krule)$
shows $s \sim d \sim s'$
using *p2 update-lsm-rule-def security-audit-rule-known-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *update-lsm-rule-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *AuditEvt*(*Event-audit-rule-known pid krule*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-aduit-evevt-def* **by** *auto*
have *a1*: $s' = fst(update\text{-}lsm\text{-}rule\ s\ pid\ krule)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg(interference\ pid\ s\ d)$
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 update-lsm-rule-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *update-lsm-rule-dlocal-rsp-e*: *dynamic-local-respect-e*(*AuditEvt*(*Event-audit-rule-known pid krule*))
using *update-lsm-rule-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.29.4 proving "audit$_rule_match$" satisfyingthe" localrespect" property

lemma *audit-rule-match-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*audit-rule-match s pid sid*)
shows $s \sim d \sim s'$
using *p2 audit-rule-match-def security-audit-rule-match-def*
by (*metis fstI kvpeq-reflexive-lemma*)

lemma *audit-rule-match-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: e = *AuditEvt*(*Event-audit-rule-match pid sid*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: s' = *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-aduit-evevt-def* **by** *auto*
have *a1*: $s' = fst$(*audit-rule-match s pid sid*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 audit-rule-match-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *audit-rule-match-dlocal-rsp-e*: *dynamic-local-respect-e*(*AuditEvt*(*Event-audit-rule-match pid sid*))
using *audit-rule-match-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.29.5 proving "audit$_free_lsm_field$" satisfyingthe" localrespect" property

lemma *audit-free-lsm-field-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)

and $p2$: $s' =$ $fst(audit\text{-}free\text{-}lsm\text{-}field\ s\ pid\ f)$
shows $s \sim d \sim s'$
using *p2 audit-free-lsm-field-def kvpeq-reflexive-lemma*
by *auto*

lemma *audit-free-lsm-field-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ $AuditEvt($ $Event\text{-}audit\text{-}rule\text{-}free\ pid\ f)$
and *p2*:*non-interference* $(the(domain\text{-}of\text{-}event\ e))\ s\ d$
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p1 domain-of-event-def getpid-from-aduit-evevt-def* by *auto*
have *a1*: $s' = fst(audit\text{-}free\text{-}lsm\text{-}field\ s\ pid\ f)$
using *p1 p3 exec-event-def* by *auto*
have *a2*: $\neg(interference\ pid\ s\ d)$
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 audit-free-lsm-field-local-rsp* by *blast*
}
then show *?thesis*
by *fast*
qed

lemma *audit-free-lsm-field-dlocal-rsp-e*: *dynamic-local-respect-e*$(AuditEvt($ $Event\text{-}audit\text{-}rule\text{-}free$ $pid\ f))$
using *audit-free-lsm-field-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.30 smack socket hooks local respect proof

29.30.1 proving "unix$_{s}tream_{c}onnect$"$satisfyingthe$"$localrespect$"$property$

lemma *unix-stream-connect-local-rsp*:
assumes *p0*: *reachable0 s*
and $p1$: $\neg(interference\ pid\ s\ d)$
and $p2$: $s' =$ $fst(unix\text{-}stream\text{-}connect\ s\ pid\ sock\ uaddr\ addr\text{-}len\ flags')$
shows $s \sim d \sim s'$
using *p2 unix-stream-connect-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *unix-stream-connect-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: e = $SocketEvt(Event\text{-}unix\text{-}stream\text{-}connect\ pid\ sock\ uaddr\ addr\text{-}len$

flags')
  and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
  and *p3*: *s'* = *exec-event s e*
 shows *s* ∼ *d* ∼ *s'*
  proof −
 {
  have *a0*: (*the* (*domain-of-event e*)) = *pid*
   using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
  have *a1*: *s'* = *fst*(*unix-stream-connect s pid sock uaddr addr-len flags'*)
   using *p1 p3 exec-event-def* **by** *auto*
  have *a2*: ¬(*interference pid s d*)
   using *p2 a0 non-interference-def*
   by *blast*
  have *a3*: *s* ∼ *d* ∼ *s'*
   using *a1 a2 p0 unix-stream-connect-local-rsp* **by** *blast*
 }
 then show *?thesis*
  by *fast*
qed

lemma *unix-stream-connect-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-unix-stream-connect pid sock uaddr addr-len flags'*))
 using *unix-stream-connect-local-rsp-e dynamic-local-respect-e-def non-interference-def*

 by *presburger*

29.30.2 proving "unix$_dgram_connect$" $satisfyingthe$" $localrespect$" $property$

lemma *unix-dgram-connect-local-rsp*:
 assumes *p0*: *reachable0 s*
  and *p1*: ¬(*interference pid s d*)
  and *p2*: *s'* = *fst*(*unix-dgram-connect s pid sock uaddr alen flags'*)
 shows *s* ∼ *d* ∼ *s'*
 using *p2 unix-dgram-connect-def security-unix-may-send-def*
proof −
 have *fst* (*unix-dgram-connect s pid sock uaddr alen flags'*) = *s*
  using *unix-dgram-connect-def* **by** *auto*
 then show *?thesis*
  by (*metis p2 kvpeq-reflexive-lemma*)
qed

lemma *unix-dgram-connect-local-rsp-e*:
 assumes *p0* :*reachable0 s*
  and *p1*: *e* = *SocketEvt*(*Event-unix-dgram-connect pid sock uaddr alen flags'*
)
  and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
  and *p3*: *s'* = *exec-event s e*
 shows *s* ∼ *d* ∼ *s'*

proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
have *a1*: *s'* = *fst*(*unix-dgram-connect s pid sock uaddr alen flags'*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s'*
using *a1 a2 p0 unix-dgram-connect-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *unix-dgram-connect-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-unix-dgram-connect pid sock uaddr alen flags'*))
using *unix-dgram-connect-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *presburger*

29.30.3 proving "unix$_{d}gram_{s}endmsg$"$satisfyingthe$"$localrespect$"$property$

lemma *unix-dgram-sendmsg-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s'* = *fst*(*unix-dgram-sendmsg s pid sock uaddr alen*)
shows *s* ∼ *d* ∼ *s'*
using *p2 unix-dgram-sendmsg-def*
by (*metis fst-conv kvpeq-reflexive-lemma*)

lemma *unix-dgram-sendmsg-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SocketEvt*(*Event-unix-dgram-sendmsg pid sock uaddr alen*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* ∼ *d* ∼ *s'*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
have *a1*: *s'* = *fst*(*unix-dgram-sendmsg s pid sock uaddr alen*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s'*
using *a1 a2 p0 unix-dgram-sendmsg-local-rsp* **by** *blast*

}
then show *?thesis*
by *fast*
qed

lemma *unix-dgram-sendmsg-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-unix-dgram-sendmsg pid sock uaddr alen*))
using *unix-dgram-sendmsg-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.30.4 proving "sys$_b$ind′"satisfyingthe"localrespect"property

lemma *sys-bind′-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s′* = *fst*(*sys-bind′ s pid fd umyaddr addrlen*)
shows *s* ∼ *d* ∼ *s′*
using *p2 sys-bind′-def security-socket-bind-def*
proof −
show *?thesis*
by (*metis* (*lifting*) *fstI p2 sys-bind′-def kvpeq-reflexive-lemma*)
qed

lemma *sys-bind′-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SocketEvt*(*Event-sys-bind′ pid fd umyaddr addrlen*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s′* = *exec-event s e*
shows *s* ∼ *d* ∼ *s′*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
have *a1*: *s′* = *fst*(*sys-bind′ s pid fd umyaddr addrlen*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s′*
using *a1 a2 p0 sys-bind′-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *sys-bind′-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-sys-bind′ pid fd umyaddr addrlen*))

using *sys-bind′-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.30.5 proving "sys$_c$*onnect*′"*satisfyingthe*"*localrespect*"*property*

lemma *sys-connect′-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s′* = *fst*(*sys-connect′ s pid fd uservaddr addrlen*)
shows *s* ∼ *d* ∼ *s′*
using *p2 sys-connect′-def security-socket-connect-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *sys-connect′-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SocketEvt*(*Event-sys-connect′ pid fd uservaddr addrlen*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s′* = *exec-event s e*
shows *s* ∼ *d* ∼ *s′*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
have *a1*: *s′* = *fst*(*sys-connect′ s pid fd uservaddr addrlen*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s′*
using *a1 a2 p0 sys-connect′-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *sys-connect′-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-sys-connect′ pid fd uservaddr addrlen*))
using *sys-connect′-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.30.6 proving "sock$_s$*endmsg*"*satisfyingthe*"*localrespect*"*property*

lemma *sock-sendmsg-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s′* = *fst*(*sock-sendmsg s pid sock msg*)
shows *s* ∼ *d* ∼ *s′*
using *p2 sock-sendmsg-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *sock-sendmsg-local-rsp-e*:
  assumes *p0* :*reachable0 s*
    and *p1*: *e* = *SocketEvt*(*Event-sock-sendmsg pid sock msg*)
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: *s'* = *exec-event s e*
  shows $s \sim d \sim s'$
    proof −
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
    have *a1*: *s'* = *fst*(*sock-sendmsg s pid sock msg*)
      using *p1 p3 exec-event-def* **by** *auto*
    have *a2*: ¬(*interference pid s d*)
      using *p2 a0 non-interference-def*
      by *blast*
    have *a3*: $s \sim d \sim s'$
      using *a1 a2 p0 sock-sendmsg-local-rsp* **by** *blast*
  }
  then show *?thesis*
    by *fast*
qed

lemma *sock-sendmsg-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-sock-sendmsg pid sock msg*))
  using *sock-sendmsg-local-rsp-e dynamic-local-respect-e-def non-interference-def*
  by *blast*

29.30.7 proving "sock$_r$*ecvmsg*"*satisfyingthe*"*localrespect*"*property*

lemma *sock-recvmsg-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: ¬(*interference pid s d*)
    and *p2*: *s'* = *fst*(*sock-recvmsg s pid sock msg flags'*)
  shows $s \sim d \sim s'$
  using *p2 sock-recvmsg-def*
 by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *sock-recvmsg-local-rsp-e*:
  assumes *p0* :*reachable0 s*
    and *p1*: *e* = *SocketEvt*(*Event-sock-recvmsg pid sock msg flags'*)
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: *s'* = *exec-event s e*
  shows $s \sim d \sim s'$
    proof −
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
    have *a1*: *s'* = *fst*(*sock-recvmsg s pid sock msg flags'*)

using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 sock-recvmsg-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *sock-recvmsg-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-sock-recvmsg pid sock msg flags'*))
using *sock-recvmsg-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.30.8 proving "sk$_f$$ilter_t rim_c ap$"$satisfyingthe$"$localrespect$"$property$

lemma *sk-filter-trim-cap-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s'* = *fst*(*sk-filter-trim-cap s pid sk' skb cap*)
shows $s \sim d \sim s'$
using *p2 sk-filter-trim-cap-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *sk-filter-trim-cap-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SocketEvt*(*Event-sk-filter-trim-cap pid sk' skb cap*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
have *a1*: *s'* = *fst*(*sk-filter-trim-cap s pid sk' skb cap*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 sk-filter-trim-cap-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *sk-filter-trim-cap-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-sk-filter-trim-cap*

pid sk′ skb cap))
using *sk-filter-trim-cap-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.30.9 proving "sock$_getsockopt$"$satisfyingthe$"$localrespect$"$property$

lemma *sock-getsockopt-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s′* = *fst*(*sock-getsockopt s pid sock level′ optname optval optlen*)
shows *s* ∼ *d* ∼ *s′*
proof −
show *?thesis*
by (*metis fst-conv p2 sock-getsockopt-def kvpeq-reflexive-lemma*)
qed

lemma *sock-getsockopt-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SocketEvt*(*Event-sock-getsockopt pid sock level′ optname optval optlen*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s′* = *exec-event s e*
shows *s* ∼ *d* ∼ *s′*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
have *a1*: *s′* = *fst*(*sock-getsockopt s pid sock level′ optname optval optlen*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s′*
using *a1 a2 p0 sock-getsockopt-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *sock-getsockopt-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-sock-getsockopt pid sock level′ optname optval optlen*))
using *sock-getsockopt-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *presburger*

29.30.10 proving "unix$_get_peersec_dgram$"$satisfyingthe$"$localrespect$"$property$

lemma *unix-get-peersec-dgram-local-rsp*:

assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*unix-get-peersec-dgram s pid sock scm*)
shows $s \sim d \sim s'$
using *p2 unix-get-peersec-dgram-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *unix-get-peersec-dgram-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *SocketEvt*(*Event-unix-get-peersec-dgram pid sock scm*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-socket-evevt-def* **by** *auto*
have *a1*: $s' = fst$(*unix-get-peersec-dgram s pid sock scm*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 unix-get-peersec-dgram-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *unix-get-peersec-dgram-dlocal-rsp-e*: *dynamic-local-respect-e*(*SocketEvt*(*Event-unix-get-peersec-dgram pid sock scm*))
using *unix-get-peersec-dgram-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *blast*

29.31 inode local respect

29.31.1 proving "vfs$_l$*ink*"*satisfyingthe*"*localrespect*"*property*

lemma *vfs-link-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*vfs-link s pid old-dentry dir new-dentry delegated-inode*)
shows $s \sim d \sim s'$
using *p2 vfs-link-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *vfs-link-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *InodeEvt*(*Event-vfs-link pid old-dentry dir new-dentry delegated-inode*

)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
have *a1*: $s' =$ *fst*(*vfs-link s pid old-dentry dir new-dentry delegated-inode*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 vfs-link-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *vfs-link-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-vfs-link pid old-dentry dir new-dentry delegated-inode*))
using *vfs-link-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *presburger*

29.31.2 proving "vfs$_unlink" satisfying the " local respect" property$

lemma *vfs-unlink-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: $s' =$ *fst*(*vfs-unlink s pid dir dentry delegated-inode*)
shows $s \sim d \sim s'$
using *p2 vfs-unlink-def*
by (*simp add*: *kvpeq-reflexive-lemma*)

lemma *vfs-unlink-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *InodeEvt*(*Event-vfs-unlink pid dir dentry delegated-inode*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
have *a1*: $s' =$ *fst*(*vfs-unlink s pid dir dentry delegated-inode*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*

by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 vfs-unlink-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *vfs-unlink-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-vfs-unlink pid dir dentry delegated-inode*))
using *vfs-unlink-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *blast*

29.31.3 proving "$\mathbf{vfs}_r mdir" satisfying the "local respect" property$

lemma *vfs-rmdir-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' =$ *fst*(*vfs-rmdir s pid dir dentry*)
shows $s \sim d \sim s'$
using *p2 vfs-rmdir-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *vfs-rmdir-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e =$ *InodeEvt*(*Event-vfs-rmdir pid dir dentry*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof $-$
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
have *a1*: $s' =$ *fst*(*vfs-rmdir s pid dir dentry*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 vfs-rmdir-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *vfs-rmdir-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-vfs-rmdir pid dir dentry*))
using *vfs-rmdir-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *presburger*

29.31.4 proving "vfs$_rename$" $satisfyingthe$" $localrespect$" $property$

lemma *vfs-rename-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*vfs-rename s pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*)
shows $s \sim d \sim s'$
using *p2 vfs-rename-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *vfs-rename-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *InodeEvt*(*Event-vfs-rename pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' =$ *exec-event s e*
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
have *a1*: $s' =$ *fst*(*vfs-rename s pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 vfs-rename-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *vfs-rename-dlocal-rsp-e*: *dynamic-local-respect-e*
(*InodeEvt*(*Event-vfs-rename pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*))
using *vfs-rename-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *presburger*

29.31.5 proving "inode$_permission$" $satisfyingthe$" $localrespect$" $property$

lemma *inode-permission-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst$(*inode-permission s pid inode mask'*)
shows $s \sim d \sim s'$
using *p2 inode-permission-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *inode-permission-local-rsp-e*:
  assumes *p0* :*reachable0 s*
    and *p1*: e = *InodeEvt*(*Event-inode-permission pid inode mask'*)
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: s' = *exec-event s e*
  shows $s \sim d \sim s'$
    proof –
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
    have *a1*: s' = *fst*(*inode-permission s pid inode mask'*)
      using *p1 p3 exec-event-def* **by** *auto*
    have *a2*: ¬(*interference pid s d*)
      using *p2 a0 non-interference-def*
      by *blast*
    have *a3*: $s \sim d \sim s'$
      using *a1 a2 p0 inode-permission-local-rsp* **by** *blast*
  }
  then show *?thesis*
    by *fast*
qed

lemma *inode-permission-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-inode-permission pid inode mask'*))
  using *inode-permission-local-rsp-e dynamic-local-respect-e-def non-interference-def*

  by *presburger*

29.31.6 proving "notify$_change"satisfyingthe"localrespect"property$

lemma *notify-change-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: ¬(*interference pid s d*)
    and *p2*: s' = *fst*(*notify-change s pid dentry attr' delegated-inode*)
  shows $s \sim d \sim s'$
proof –
  show *?thesis*
    by (*metis fstI notify-change-def p2 kvpeq-reflexive-lemma*)
qed

lemma *notify-change-local-rsp-e*:
  assumes *p0* :*reachable0 s*
    and *p1*: e = *InodeEvt*(*Event-notify-change pid dentry attr' delegated-inode*)
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: s' = *exec-event s e*
  shows $s \sim d \sim s'$
    proof –

{
  have *a0*: (*the* (*domain-of-event e*)) = *pid*
    using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
  have *a1*: *s′* = *fst*(*notify-change s pid dentry attr′ delegated-inode*)
    using *p1 p3 exec-event-def* **by** *auto*
  have *a2*: ¬(*interference pid s d*)
    using *p2 a0 non-interference-def*
    by *blast*
  have *a3*: *s* ∼ *d* ∼ *s′*
    using *a1 a2 p0 notify-change-local-rsp* **by** *blast*
}
then show *?thesis*
  by *fast*
qed

lemma *notify-change-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-notify-change pid dentry attr′ delegated-inode*))
  using *notify-change-local-rsp-e dynamic-local-respect-e-def non-interference-def*
  by *presburger*

29.31.7 proving "fat$_i$$octl_set_attributes$"$satisfyingthe$"$localrespect$"$property$

lemma *fat-ioctl-set-attributes-local-rsp*:
  assumes *p0*: *reachable0 s*
    and *p1*: ¬(*interference pid s d*)
    and *p2*: *s′* = *fst*(*fat-ioctl-set-attributes s pid f*)
  shows *s* ∼ *d* ∼ *s′*
  using *p2*
  unfolding *fat-ioctl-set-attributes-def*
  by (*metis fstI kvpeq-reflexive-lemma*)

lemma *fat-ioctl-set-attributes-local-rsp-e*:
  assumes *p0* :*reachable0 s*
    and *p1*: *e* = *InodeEvt*(*Event-fat-ioctl-set-attributes pid f*)
    and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
    and *p3*: *s′* = *exec-event s e*
  shows *s* ∼ *d* ∼ *s′*
    proof −
  {
    have *a0*: (*the* (*domain-of-event e*)) = *pid*
      using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
    have *a1*: *s′* = *fst*(*fat-ioctl-set-attributes s pid f*)
      using *p1 p3 exec-event-def* **by** *auto*
    have *a2*: ¬(*interference pid s d*)
      using *p2 a0 non-interference-def*
      by *blast*
    have *a3*: *s* ∼ *d* ∼ *s′*
      using *a1 a2 p0 fat-ioctl-set-attributes-local-rsp* **by** *blast*

}
then show *?thesis*
by *fast*
qed

lemma *fat-ioctl-set-attributes-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-fat-ioctl-set-attributes pid f*))
using *fat-ioctl-set-attributes-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *presburger*

29.31.8 proving "vfs$_getattr" satisfyingthe" localrespect" property$

lemma *vfs-getattr-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: $s' = fst(vfs\text{-}getattr\ s\ pid\ path)$
shows $s \sim d \sim s'$
using *p2 vfs-getattr-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *vfs-getattr-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *InodeEvt*(*Event-vfs-getattr pid path*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof –
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
have *a1*: $s' = fst(vfs\text{-}getattr\ s\ pid\ path)$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 vfs-getattr-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *vfs-getattr-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-vfs-getattr pid path*))
using *vfs-getattr-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *presburger*

29.31.9 proving "vfs$_s$*etxattr*" *satisfyingthe*" *localrespect*" *property*

lemma *vfs-setxattr-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: *s' = fst*(*vfs-setxattr s pid dentry name value size' flgs*)
shows *s* $\sim$ *d* $\sim$ *s'*
using *p2 vfs-setxattr-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *vfs-setxattr-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *InodeEvt*(*Event-vfs-setxattr pid dentry name value size' flgs*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* $\sim$ *d* $\sim$ *s'*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
have *a1*: *s'* = *fst*(*vfs-setxattr s pid dentry name value size' flgs*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* $\sim$ *d* $\sim$ *s'*
using *a1 a2 p0 vfs-setxattr-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *vfs-setxattr-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-vfs-setxattr pid dentry name value size' flgs*))
using *vfs-setxattr-local-rsp-e dynamic-local-respect-e-def non-interference-def*
by *presburger*

29.31.10 proving "vfs$_g$*etxattr*" *satisfyingthe*" *localrespect*" *property*

lemma *vfs-getxattr-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: *s' = fst*(*vfs-getxattr s pid dentry name value size'*)
shows *s* $\sim$ *d* $\sim$ *s'*
using *p2 vfs-getxattr-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *vfs-getxattr-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *InodeEvt*(*Event-vfs-getxattr pid dentry name value size'*)

and *p2*:*non-interference* (*the*(*domain-of-event* *e*)) *s* *d*
and *p3*: $s' =$ *exec-event* *s* *e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event* *e*)) = *pid*
using *p1* *domain-of-event-def* *getpid-from-inode-evevt-def* by *auto*
have *a1*: $s' =$ *fst*(*vfs-getxattr* *s* *pid* *dentry* *name* *value* *size'*)
using *p1* *p3* *exec-event-def* by *auto*
have *a2*: ¬(*interference* *pid* *s* *d*)
using *p2* *a0* *non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1* *a2* *p0* *vfs-getxattr-local-rsp* by *blast*
}
then show *?thesis*
by *fast*
qed

lemma *vfs-getxattr-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-vfs-getxattr* *pid* *dentry* *name* *value* *size'*))
using *vfs-getxattr-local-rsp-e* *dynamic-local-respect-e-def* *non-interference-def*
by *presburger*

29.31.11 proving "vfs$_r$*emovexattr*"*satisfyingthe*"*localrespect*"*property*

lemma *vfs-removexattr-local-rsp*:
assumes *p0*: *reachable0* *s*
and *p1*: ¬(*interference* *pid* *s* *d*)
and *p2*: $s' =$ *fst*(*vfs-removexattr* *s* *pid* *dentry* *name*)
shows $s \sim d \sim s'$
using *p2* *vfs-removexattr-def*
by (*simp* *add*: *kvpeq-reflexive-lemma*)

lemma *vfs-removexattr-local-rsp-e*:
assumes *p0* :*reachable0* *s*
and *p1*: *e* = *InodeEvt*(*Event-vfs-removexattr* *pid* *dentry* *name*)
and *p2*:*non-interference* (*the*(*domain-of-event* *e*)) *s* *d*
and *p3*: $s' =$ *exec-event* *s* *e*
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event* *e*)) = *pid*
using *p1* *domain-of-event-def* *getpid-from-inode-evevt-def* by *auto*
have *a1*: $s' =$ *fst*(*vfs-removexattr* *s* *pid* *dentry* *name*)
using *p1* *p3* *exec-event-def* by *auto*
have *a2*: ¬(*interference* *pid* *s* *d*)
using *p2* *a0* *non-interference-def*
by *blast*

have *a3*: *s* ∼ *d* ∼ *s'*
using *a1 a2 p0 vfs-removexattr-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *vfs-removexattr-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-vfs-removexattr pid dentry name*))
using *vfs-removexattr-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *presburger*

29.31.12 proving "xattr$_g$etsecurity"satisfyingthe"localrespect"property

lemma *xattr-getsecurity-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: ¬(*interference pid s d*)
and *p2*: *s'* = *fst*(*xattr-getsecurity s pid inode name value size'*)
shows *s* ∼ *d* ∼ *s'*
using *p2 xattr-getsecurity-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *xattr-getsecurity-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *InodeEvt*(*Event-xattr-getsecurity pid inode name value size'*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* ∼ *d* ∼ *s'*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
have *a1*: *s'* = *fst*(*xattr-getsecurity s pid inode name value size'*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* ∼ *d* ∼ *s'*
using *a1 a2 p0 xattr-getsecurity-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *xattr-getsecurity-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-xattr-getsecurity pid inode name value size'*))
using *xattr-getsecurity-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *presburger*

29.31.13 proving "nfs4$_listxattr_nfs4_label$" satisfying the "local respect" property

lemma *nfs4-listxattr-nfs4-label-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: *s'* = *fst*(*nfs4-listxattr-nfs4-label s pid inode name size'*)
shows *s* $\sim$ *d* $\sim$ *s'*
using *p2 nfs4-listxattr-nfs4-label-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *nfs4-listxattr-nfs4-label-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: *e* = *InodeEvt*(*Event-nfs4-listxattr-nfs4-label pid inode name size'*)
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: *s'* = *exec-event s e*
shows *s* $\sim$ *d* $\sim$ *s'*
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
have *a1*: *s'* = *fst*(*nfs4-listxattr-nfs4-label s pid inode name size'*)
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: $\neg$(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: *s* $\sim$ *d* $\sim$ *s'*
using *a1 a2 p0 nfs4-listxattr-nfs4-label-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *nfs4-listxattr-nfs4-label-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-nfs4-listxattr-nfs4-label pid inode name size'*))
using *nfs4-listxattr-nfs4-label-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *presburger*

29.31.14 proving "sockfs$_listxattr$" satisfying the "local respect" property

lemma *sockfs-listxattr-local-rsp*:
assumes *p0*: *reachable0 s*
and *p1*: $\neg$(*interference pid s d*)
and *p2*: *s'* = *fst*(*sockfs-listxattr s pid dentry buffer size'*)
shows *s* $\sim$ *d* $\sim$ *s'*
using *p2 sockfs-listxattr-def*
by (*smt fst-conv kvpeq-reflexive-lemma*)

lemma *sockfs-listxattr-local-rsp-e*:
assumes *p0* :*reachable0 s*
and *p1*: $e = InodeEvt(\ Event\text{-}sockfs\text{-}listxattr\ pid\ \ dentry\ buffer\ \ size')$
and *p2*:*non-interference* (*the*(*domain-of-event e*)) *s d*
and *p3*: $s' = exec\text{-}event\ s\ e$
shows $s \sim d \sim s'$
proof −
{
have *a0*: (*the* (*domain-of-event e*)) = *pid*
using *p1 domain-of-event-def getpid-from-inode-evevt-def* **by** *auto*
have *a1*: $s' = fst(sockfs\text{-}listxattr\ s\ pid\ \ dentry\ buffer\ \ size')$
using *p1 p3 exec-event-def* **by** *auto*
have *a2*: ¬(*interference pid s d*)
using *p2 a0 non-interference-def*
by *blast*
have *a3*: $s \sim d \sim s'$
using *a1 a2 p0 sockfs-listxattr-local-rsp* **by** *blast*
}
then show *?thesis*
by *fast*
qed

lemma *sockfs-listxattr-dlocal-rsp-e*: *dynamic-local-respect-e*(*InodeEvt*(*Event-sockfs-listxattr pid dentry buffer size'*))
using *sockfs-listxattr-local-rsp-e dynamic-local-respect-e-def non-interference-def*

by *presburger*

29.31.15 proving the "dynamic local respect" property

definition *dynamic-local-respect-c* :: *bool* **where**
dynamic-local-respect-c ≡ ∀ *e d s*. *reachable0 s*
∧ ¬(*interference* (*the* (*domain-of-event e*)) *s d*)
⟶ ($s \sim d \sim$ (*exec-event s e*))

thm *Event-tsk.induct*

theorem *dynamic-local-respect*:*dynamic-local-respect*
proof −
{
fix *e*
have *dynamic-local-respect-e e*
apply(*induct e*)
using *k-sb-copy-data-dlocal-rsp-e do-remount-dlocal-rsp-e mount-fs-dlocal-rsp-e k-show-sb-opts-dlocal-rsp-e*
sb-statfs-dlocal-rsp-e do-mount-dlocal-rsp-e do-umount-dlocal-rsp-e
pivot-root-dlocal-rsp-e setup-security-options-dlocal-rsp-e set-sb-security-dlocal-rsp-e
nfs-clone-sb-security-dlocal-rsp-e parse-security-options-dlocal-rsp-e
apply (*metis Event-sb.exhaust*)

using *prepare-creds-dlocal-rsp-e sys-setreuid-dlocal-rsp-e setpgid-dlocal-rsp-e*

do-getpgid-dlocal-rsp-e getsid-dlocal-rsp-e getsecid-dlocal-rsp-e
task-setnice-dlocal-rsp-e set-task-ioprio-dlocal-rsp-e
get-task-ioprio-dlocal-rsp-e check-prlimit-permission-dlocal-rsp-e
do-prlimit-dlocal-rsp-e task-setscheduler-dlocal-rsp-e
task-getscheduler-dlocal-rsp-e task-movememory-dlocal-rsp-e
task-kill-dlocal-rsp-e task-prctl-dlocal-rsp-e
apply (*metis Event-tsk.exhaust*)
using *ptrace-may-access-dlocal-rsp-e ptrace-traceme-dlocal-rsp-e Event-Ptrace.exhaust*
apply *metis*
using *check-syslog-permissions-dlocal-rsp-e prepare-binprm-dlocal-rsp-e*
apply (*metis Event-sys.exhaust*)
using *do-ioctl-dlocal-rsp-e syscall-ioctl-dlocal-rsp-e*
ksys-ioctl-dlocal-rsp-e vm-mmap-pgoff-dlocal-rsp-e do-sys-vm86-dlocal-rsp-e
get-unmapped-area-dlocal-rsp-e
validate-mmap-request-dlocal-rsp-e generic-setlease-dlocal-rsp-e syscall-lock-dlocal-rsp-e

do-lock-file-wait-dlocal-rsp-e file-fcntl-dlocal-rsp-e
file-send-sigiotask-dlocal-rsp-e file-receive-dlocal-rsp-e do-dentry-open-dlocal-rsp-e
apply (*metis Event-file.exhaust*)
using *ipcperms-dlocal-rsp-e audit-ipc-obj-dlocal-rsp-e msg-queue-msgctl-dlocal-rsp-e*

do-msgsnd-dlocal-rsp-e msg-queue-msgrcv-dlocal-rsp-e ksys-shmget-dlocal-rsp-e

sem-msgctl-dlocal-rsp-e do-semtimedop-dlocal-rsp-e do-shmat-dlocal-rsp-e

ksys-semget-dlocal-rsp-e shm-msgctl-dlocal-rsp-e ksys-msgget-dlocal-rsp-e
apply (*metis Event-ipc.exhaust*)
using *key-task-permission-dlocal-rsp-e keyctl-get-security-dlocal-rsp-e*
apply (*metis Event-Key.exhaust*)
using *audit-data-to-entry-dlocal-rsp-e audit-dupe-lsm-field-dlocal-rsp-e*
update-lsm-rule-dlocal-rsp-e
audit-rule-match-dlocal-rsp-e audit-free-lsm-field-dlocal-rsp-e
using *Event-audit.exhaust* **apply** *metis*
using *unix-stream-connect-dlocal-rsp-e unix-dgram-connect-dlocal-rsp-e*
unix-dgram-sendmsg-dlocal-rsp-e sys-bind'-dlocal-rsp-e
sys-connect'-dlocal-rsp-e sock-sendmsg-dlocal-rsp-e
sock-recvmsg-dlocal-rsp-e sk-filter-trim-cap-dlocal-rsp-e
sock-getsockopt-dlocal-rsp-e unix-get-peersec-dgram-dlocal-rsp-e
apply (*metis Event-network-sock.exhaust*)
using *vfs-link-dlocal-rsp-e vfs-unlink-dlocal-rsp-e vfs-rmdir-dlocal-rsp-e*
vfs-rename-dlocal-rsp-e
inode-permission-dlocal-rsp-e notify-change-dlocal-rsp-e fat-ioctl-set-attributes-dlocal-rsp-e

vfs-getattr-dlocal-rsp-e vfs-setxattr-dlocal-rsp-e vfs-getxattr-dlocal-rsp-e
vfs-removexattr-dlocal-rsp-e xattr-getsecurity-dlocal-rsp-e
nfs4-listxattr-nfs4-label-dlocal-rsp-e sockfs-listxattr-dlocal-rsp-e
by (*metis Event-inode.exhaust*)

}
then show *?thesis*
using *dynamic-local-respect-all-evt* **by** *blast*
qed

29.32 Concrete unwinding condition of "weakly step consistent"

29.33 smack super$_{b}lockhooksweaklystepconsistent$

29.33.1 proving "sb$_{c}opy_{d}ata$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *sb-copy-data-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst$ (*k-sb-copy-data s pid*)
and *p6*: $t' = fst$ (*k-sb-copy-data t pid*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 k-sb-copy-data-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 k-sb-copy-data-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sb-copy-data-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SbEvt* (*Event-sb-copy-data pid sb*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*

by *force*
have *a1*: *s′* = *fst* (*k-sb-copy-data s pid*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t′* = *fst* (*k-sb-copy-data t pid*)
using *p2 p7 exec-event-def* **by** *auto*
have *a3*: *pid* @ *s d*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s′* ∼ *d* ∼ *t′*
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 sb-copy-data-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed
lemma *sb-copy-data-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-copy-data pid sb*))
proof −
{
have ∀ *d s t*. (*reachable0 s*) ∧ (*reachable0 t*) ∧
(*s* ∼ *d* ∼ *t*) ∧
((*the* (*domain-of-event* (*SbEvt* (*Event-sb-copy-data pid sb*)))) @ *s d*)∧
(*s* ∼ (*the* (*domain-of-event* (*SbEvt* (*Event-sb-copy-data pid sb*)))) ∼ *t*) −→
((*exec-event s* (*SbEvt* (*Event-sb-copy-data pid sb*))) ∼ *d* ∼ (*exec-event t* (*SbEvt* (*Event-sb-copy-data pid sb*))))
proof −
{
fix *d s t*
let *?e* = *SbEvt* (*Event-sb-copy-data pid sb*)
assume *p2*: *reachable0 s*
assume *p3*: *reachable0 t*
assume *p4*: (*s* ∼ *d* ∼ *t*)
assume *p5*: (*the* (*domain-of-event ?e*)) @ *s d*
assume *p6*: (*s* ∼ (*the* (*domain-of-event ?e*)) ∼ *t*)
have *a0*: (*the* (*domain-of-event ?e*)) = *pid*
using *domain-of-event-def getpid-from-sb-event-def*
by *auto*
have (*exec-event s ?e*) ∼ *d* ∼ (*exec-event t ?e*)
using *p2 p3 p4 p5 p6 sb-copy-data-wsc-e*
by *blast*
}
then show *?thesis* **by** *blast*
qed
}
then show *?thesis*
using *dynamic-weakly-step-consistent-e-def* **by** *blast*
qed

29.33.2 proving "do$_remount" satisfying the "weakly step consistent" property$

lemma *do-remount-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(do\text{-}remount\ s\ p\ v)$
and *p6*: $t' = fst(do\text{-}remount\ t\ p\ v)$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 do-remount-def*
by *simp*
have *a1* : $t = t'$
using *p6 do-remount-def*
by *simp*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2* **by** *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-remount-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = SbEvt\ (Event\text{-}sb\text{-}remount\ pid\ p\ v)$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*
by *force*
have *a1*: $s' = fst(do\text{-}remount\ s\ p\ v)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(do\text{-}remount\ t\ p\ v)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*

have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-remount-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed
lemma *do-remount-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-remount pid p v*))
using *dynamic-weakly-step-consistent-e-def do-remount-wsc-e* **by** *blast*

29.33.3 proving "mount$_{f}s$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *mount-fs-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*mount-fs s ts f name data*)
and *p6*: $t' =$ *fst*(*mount-fs t ts f name data*)
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 mount-fs-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 mount-fs-def*
by (*smt fstI*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2* **by** *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *mount-fs-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e =$ *SbEvt* (*Event-sb-kern-mount pid ts f name data*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*

by *force*
have *a1*: $s' = fst(mount\text{-}fs\ s\ ts\ f\ name\ data\ )$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(mount\text{-}fs\ t\ ts\ f\ name\ data\ )$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 mount-fs-wsc*
by *fast*

}
then show *?thesis* **by** *auto*
qed
lemma *mount-fs-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-kern-mount pid t f name data*))
using *dynamic-weakly-step-consistent-e-def mount-fs-wsc-e* **by** *blast*

29.33.4 proving "show$_{s}b_{o}pts$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *show-sb-opts-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(show\text{-}sb\text{-}opts\ s\ pid\ sq\ sb)$
and *p6*: $t' = fst(show\text{-}sb\text{-}opts\ t\ pid\ sq\ sb)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 show-sb-opts-def*
by *simp*
have *a1* : $t = t'$
using *p6 show-sb-opts-def*
by *simp*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2* **by** *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *show-sb-opts-wsc-e*:
assumes *p0*: *reachable0 s*

and $p1$: *reachable0 t*
and $p2$: e = *SbEvt* (*Event-sb-show-options pid sq sb*)
and $p3$: $s \sim d \sim t$
and $p4$: (*the* (*domain-of-event e*)) @ s d
and $p5$: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and $p6$: s' = *exec-event s e*
and $p7$: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*
by *force*
have *a1*: s' = *fst*(*show-sb-opts s pid sq sb*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*show-sb-opts t pid sq sb*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 show-sb-opts-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed
lemma *show-sb-opts-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-show-options pid sq sb*))
using *dynamic-weakly-step-consistent-e-def show-sb-opts-wsc-e* **by** *blast*

29.33.5 proving "statfs$_{b}y_{d}entry$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *statfs-by-dentry-wsc*:
assumes $p0$: *reachable0 s*
and $p1$: *reachable0 t*
and $p2$: $s \sim d \sim t$
and $p3$: *pid* @ s d
and $p4$: $s \sim pid \sim t$
and $p5$: s' = *fst*(*statfs-by-dentry s pid de*)
and $p6$: t' = *fst*(*statfs-by-dentry t pid de*)
shows $s' \sim d \sim t'$
using *p6 p5 p2 statfs-by-dentry-def*
by (*metis fst-conv*)

lemma *statfs-by-dentry-wsc-e*:

assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SbEvt* (*Event-sb-statfs pid de*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s′* = *exec-event s e*
and *p7*: *t′* = *exec-event t e*
shows *s′* ∼ *d* ∼ *t′*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*
by *force*
have *a1*: *s′* = *fst*(*statfs-by-dentry s pid de*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t′* = *fst*(*statfs-by-dentry t pid de*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s′* ∼ *d* ∼ *t′*
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 statfs-by-dentry-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *statfs-by-dentry-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-statfs pid d*))
using *dynamic-weakly-step-consistent-e-def statfs-by-dentry-wsc-e*
by *blast*

29.33.6 proving "do$_mount$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *do-mount-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s′* = *fst*(*do-mount s pid devname dirname tp f p*)
and *p6*: *t′* = *fst*(*do-mount t pid devname dirname tp f p*)
shows *s′* ∼ *d* ∼ *t′*
using *p6 p5 p2 do-mount-def fst-conv*
by (*metis*)

lemma *do-mount-wsc-e*:
  assumes *p0*: *reachable0 s*
    and *p1*: *reachable0 t*
    and *p2*: $e = SbEvt$ (*Event-sb-mount pid devname dirname tp f p*)
    and *p3*: $s \sim d \sim t$
    and *p4*: (*the* (*domain-of-event e*)) @ *s d*
    and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
    and *p6*: $s' =$ *exec-event s e*
    and *p7*: $t' =$ *exec-event t e*
  shows $s' \sim d \sim t'$
  proof –
    {
    have *a0* : (*the* (*domain-of-event e*)) = *pid*
      using *p2 domain-of-event-def getpid-from-sb-event-def*
      by *force*
    have *a1*: $s' =$ *fst*(*do-mount s pid devname dirname tp f p*)
      using *p2 p6 exec-event-def* **by** *auto*
    have *a2*: $t' =$ *fst*(*do-mount t pid devname dirname tp f p*)
      using *p2 p7 exec-event-def*
      by *auto*
    have *a3*: *pid* @ *sd*
      using *p4 a0*
      by *blast*
    have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
      by *blast*
    have *a5*: $s' \sim d \sim t'$
      using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-mount-wsc*
      by *blast*
    }
  then show *?thesis* **by** *auto*
qed

lemma *do-mount-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-mount pid devname dirname tp f p*))
  using *dynamic-weakly-step-consistent-e-def do-mount-wsc-e*
  by *blast*

29.33.7 proving "do$_u$*mount*"*satisfyingthe*"*weaklystepconsistent*"*property*

lemma *do-umount-wsc*:
 assumes *p0*: *reachable0 s*
    and *p1*: *reachable0 t*
    and *p2*: $s \sim d \sim t$
    and *p3*: *pid* @ *s d*
    and *p4*: $s \sim pid \sim t$
    and *p5*: $s' =$ *fst* (*do-umount s pid m i*)
    and *p6*: $t' =$ *fst* (*do-umount t pid m i*)

shows $s' \sim d \sim t'$
using *p6 p5 p2 do-umount-def fst-conv*
by (*metis*)

lemma *do-umount-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = SbEvt\ (Event\text{-}sb\text{-}umount\ pid\ m\ i)$
and *p3*: $s \sim d \sim t$
and *p4*: $(the\ (domain\text{-}of\text{-}event\ e))\ @\ s\ d$
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p2 domain-of-event-def getpid-from-sb-event-def*
by *force*
have *a1*: $s' = fst\ (do\text{-}umount\ s\ pid\ m\ i)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst\ (do\text{-}umount\ t\ pid\ m\ i)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: $pid\ @\ sd$
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-umount-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-umount-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-umount pid m i*))
using *dynamic-weakly-step-consistent-e-def do-umount-wsc-e*
by *blast*

29.33.8 proving "pivot$_{r}oot$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *pivot-root-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: $pid\ @\ s\ d$
and *p4*: $s \sim pid \sim t$

and *p5*: $s' = fst$ (*pivot-root s pid*)
and *p6*: $t' = fst$ (*pivot-root t pid*)
shows $s' \sim d \sim t'$
using *p6 p5 p2 pivot-root-def fst-conv*
by (*metis*)

lemma *pivot-root-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SbEvt* (*Event-sb-pivotroot pid*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*
by *force*
have *a1*: $s' =$ *fst* (*pivot-root s pid*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst* (*pivot-root t pid*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 pivot-root-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *pivot-root-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-pivotroot pid*))
using *dynamic-weakly-step-consistent-e-def pivot-root-wsc-e*
by *blast*

29.33.9 proving "setup$_security_options$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *setup-security-options-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$

and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s′* = *fst*(*setup-security-options s pid n sb opt*)
and *p6*: *t′* = *fst*(*setup-security-options t pid n sb opt*)
shows *s′* ∼ *d* ∼ *t′*
using *p6 p5 p2 setup-security-options-def fst-conv*
by (*metis*)

lemma *setup-security-options-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SbEvt* (*Event-set-mnt-opts pid n sb opt*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s′* = *exec-event s e*
and *p7*: *t′* = *exec-event t e*
shows *s′* ∼ *d* ∼ *t′*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*
by *force*
have *a1*: *s′* = *fst*(*setup-security-options s pid n sb opt*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t′* = *fst*(*setup-security-options t pid n sb opt*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *s d*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s′* ∼ *d* ∼ *t′*
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 setup-security-options-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *setup-security-options-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-set-mnt-opts pid n sb opt*))
using *dynamic-weakly-step-consistent-e-def setup-security-options-wsc-e*
by *blast*

29.33.10 proving "set$_{s}b_{s}ecurity$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *set-sb-security-wsc*:
assumes *p0*: *reachable0 s*

and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(set\text{-}sb\text{-}security\ s\ pid\ sb\ de\ info)$
and *p6*: $t' = fst(set\text{-}sb\text{-}security\ t\ pid\ sb\ de\ info)$
shows $s' \sim d \sim t'$
using *p6 p5 p2 set-sb-security-def fst-conv*
by (*metis*)

lemma *set-sb-security-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SbEvt* (*Event-set-sb-security pid sb de info*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*
by *force*
have *a1*: $s' = fst(set\text{-}sb\text{-}security\ s\ pid\ sb\ de\ info)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(set\text{-}sb\text{-}security\ t\ pid\ sb\ de\ info)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 set-sb-security-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *set-sb-security-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-set-sb-security pid sb d info*))
using *dynamic-weakly-step-consistent-e-def set-sb-security-wsc-e*
by *blast*

29.33.11 proving "nfs$_{c}lone_{s}b_{s}ecurity$" $satisfying the$ " $weakly step consistent$ " $property$

lemma *nfs-clone-sb-security-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst$(*nfs-clone-sb-security s pid sb de minfo*)
and *p6*: $t' = fst$(*nfs-clone-sb-security s pid sb de minfo*)
shows $s' \sim d \sim t'$
using *p6 p5 p2 nfs-clone-sb-security-def fst-conv*
by (*simp add*: *vpeq-reflexive-lemma*)

thm *nfs-clone-sb-security-def*
lemma *nfs-clone-sb-security-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SbEvt* (*Event-sb-clone-mnt-opts pid sb de minfo*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*
by *force*
have *a1*: $s' = fst$(*nfs-clone-sb-security s pid sb de minfo*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst$(*nfs-clone-sb-security t pid sb de minfo*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: *result s* (*security-sb-clone-mnt-opts' s oldsb sb' kflags kflags-out*) = *True*
apply(*simp add*: *security-sb-clone-mnt-opts'-def result-def the-run-state-def*)

by (*simp add*: *return-def*)
have *a6*: $s = s'$ **using** *nfs-clone-sb-security-def a5*
by (*smt a1 eq-fst-iff*)
have *a7*: *result t* (*security-sb-clone-mnt-opts' t oldsb sb' kflags kflags-out*) = *True*
apply(*simp add*: *security-sb-clone-mnt-opts'-def result-def the-run-state-def*)

by (*simp add*: *return-def*)
have *a8*: $t = t'$ **using** *nfs-clone-sb-security-def a2 a7*
by (*smt eq-fst-iff*)
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 a6 a8 set-sb-security-wsc*
by *presburger*
}
then show *?thesis* **by** *auto*
qed

lemma *nfs-clone-sb-security-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-clone-mnt-opts pid sb d minfo*))
using *dynamic-weakly-step-consistent-e-def nfs-clone-sb-security-wsc-e*
by *blast*

29.33.12 proving "parse$_s$$ecurity_options$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *parse-security-options-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst$(*parse-security-options s pid str opt*)
and *p6*: $t' = fst$(*parse-security-options t pid str opt*)
shows $s' \sim d \sim t'$
using *p6 p5 p2 parse-security-options-def fst-conv*
by (*metis*)

lemma *parse-security-options-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SbEvt* (*Event-sb-parse-opts-str pid str opt*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sb-event-def*
by *force*
have *a1*: $s' = fst$(*parse-security-options s pid str opt*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst$(*parse-security-options t pid str opt*)
using *p2 p7 exec-event-def*

by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 parse-security-options-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *parse-security-options-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SbEvt* (*Event-sb-parse-opts-str pid str opt*))
using *dynamic-weakly-step-consistent-e-def parse-security-options-wsc-e*
by *blast*

29.34 smack task hooks weakly step consistent

29.34.1 proving "prepare$_c$*reds*"*satisfyingthe*"*weaklystepconsistent*"*property*

lemma *prepare-creds-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(prepare\text{-}creds\ s\ pid)$
and *p6*: $t' = fst(prepare\text{-}creds\ t\ pid)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 prepare-creds-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 prepare-creds-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *prepare-creds-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *TskEvt* ((*Event-prepare-creds pid*))

and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' =$ *fst*(*prepare-creds s pid*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*prepare-creds t pid*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 prepare-creds-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *prepare-creds-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-prepare-creds pid*)))
using *dynamic-weakly-step-consistent-e-def prepare-creds-wsc-e*
by *blast*

29.34.2 proving "sys$_{s}etreuid$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *sys-setreuid-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*sys-setreuid s pid kuid euid'*)
and *p6*: $t' =$ *fst*(*sys-setreuid t pid kuid euid'*)
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 sys-setreuid-def*
by (*smt fstI*)

have *a1* : $t = t'$
using *p6 sys-setreuid-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sys-setreuid-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = TskEvt$ ((*Event-sys-setreuid pid kuid euid'*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' =$ *fst*(*sys-setreuid s pid kuid euid'*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*sys-setreuid t pid kuid euid'*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 sys-setreuid-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sys-setreuid-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-sys-setreuid pid kuid euid'*)))
using *dynamic-weakly-step-consistent-e-def sys-setreuid-wsc-e*
by *blast*

29.34.3 proving "setpgid" satisfying the "weakly step consistent" property

lemma *setpgid-wsc*:
 assumes *p0*: *reachable0 s*
   and *p1*: *reachable0 t*
   and *p2*: $s \sim d \sim t$
   and *p3*: *pid* @ *s d*
   and *p4*: $s \sim pid \sim t$
   and *p5*: $s' = fst(setpgid\ s\ pid\ i\ pgid)$
   and *p6*: $t' = fst(setpgid\ t\ pid\ i\ pgid)$
  shows $s' \sim d \sim t'$
   proof −
  {
   have *a0* : $s = s'$
    using *p5 setpgid-def*
    by (*smt fstI*)
   have *a1* : $t = t'$
    using *p6 setpgid-def*
    by (*smt fst-conv*)
   have *a2*: $s' \sim d \sim t'$
    using *a0 a1 p2*
    by *blast*
  }
  then show *?thesis* **by** *auto*
qed

lemma *setpgid-wsc-e*:
  assumes *p0*: *reachable0 s*
   and *p1*: *reachable0 t*
   and *p2*: *e* = *TskEvt* ((*Event-setpgid pid i pgid*))
   and *p3*: $s \sim d \sim t$
   and *p4*: (*the* (*domain-of-event e*)) @ *s d*
   and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
   and *p6*: s' = *exec-event s e*
   and *p7*: t' = *exec-event t e*
  shows $s' \sim d \sim t'$
  proof −
   {
   have *a0* : (*the* (*domain-of-event e*)) = *pid*
    using *p2 domain-of-event-def getpid-from-tsk-event-def*
    by *force*
   have *a1*: $s' = fst(setpgid\ s\ pid\ i\ pgid)$
    using *p2 p6 exec-event-def* **by** *auto*
   have *a2*: $t' = fst(setpgid\ t\ pid\ i\ pgid)$
    using *p2 p7 exec-event-def*
    by *auto*
   have *a3*: *pid* @ *s d*
    using *p4 a0*
    by *blast*

have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 setpgid-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *setpgid-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-setpgid pid i pgid*)))
using *dynamic-weakly-step-consistent-e-def setpgid-wsc-e*
by *blast*

29.34.4 proving "do$_getpgid$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *do-getpgid-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(do\text{-}getpgid\ s\ pid\ i\ )$
and *p6*: $t' = fst(do\text{-}getpgid\ t\ pid\ i)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 do-getpgid-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 do-getpgid-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-getpgid-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = TskEvt$ ((*Event-do-getpgid pid i*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$

shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' = fst(do\text{-}getpgid\ s\ pid\ i)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(do\text{-}getpgid\ t\ pid\ i)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-getpgid-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-getpgid-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-do-getpgid pid i*)))
using *dynamic-weakly-step-consistent-e-def do-getpgid-wsc-e*
by *blast*

29.34.5 proving "getsid" satisfying the "weakly step consistent" property

lemma *getsid-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(getsid\ s\ pid\ i\ )$
and *p6*: $t' = fst(getsid\ t\ pid\ i)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 getsid-def*
by (*smt case-prod-unfold fstI*)
have *a1* : $t = t'$
using *p6 getsid-def*
by (*smt fstI old.prod.case*)
have *a2*: $s' \sim d \sim t'$

using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *getsid-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = $ *TskEvt* ((*Event-getsid pid i*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' =$ *fst*(*getsid s pid i*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*getsid t pid i*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *s d*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 getsid-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *getsid-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-getsid pid i*)))
using *dynamic-weakly-step-consistent-e-def getsid-wsc-e*
by *blast*

29.34.6 proving "getsecid" satisfying the "weakly step consistent" property

lemma *getsecid-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*

and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s* *d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s′* =*fst*(*getsecid s pid p u*)
and *p6*: *t′* = *fst*(*getsecid t pid p u*)
shows *s′* ∼ *d* ∼ *t′*
proof −
{
have *a0* : *s* = *s′*
using *p5 getsecid-def*
by (*smt fstI*)
have *a1* : *t* = *t′*
using *p6 getsecid-def*
by (*smt fst-conv*)
have *a2*: *s′* ∼ *d* ∼ *t′*
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *getsecid-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *TskEvt* ((*Event-getsecid pid p u*))
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s* *d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s′* = *exec-event s e*
and *p7*: *t′* = *exec-event t e*
shows *s′* ∼ *d* ∼ *t′*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: *s′* = *fst*(*getsecid s pid p u*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t′* = *fst*(*getsecid t pid p u*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s′* ∼ *d* ∼ *t′*
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 getsecid-wsc*
by *blast*

}
then show *?thesis* **by** *auto*
qed

lemma *getsecid-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-getsecid pid p u*)))
using *dynamic-weakly-step-consistent-e-def getsecid-wsc-e*
by *blast*

29.34.7 proving "task$_{s}etnice" satisfyingthe" weaklystepconsistent" property$

lemma *task-setnice-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: s' =*fst*(*task-setnice s pid p i*)
and *p6*: t' = *fst*(*task-setnice t pid p i*)
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 task-setnice-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 task-setnice-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-setnice-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *TskEvt* ((*Event-task-setnice pid p i*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*

by *force*
have *a1*: $s' = fst(task\text{-}setnice\ s\ pid\ p\ i)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(task\text{-}setnice\ t\ pid\ p\ i)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 task-setnice-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-setnice-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-task-setnice pid p i*)))
using *dynamic-weakly-step-consistent-e-def task-setnice-wsc-e*
by *blast*

29.34.8 proving "set$_t ask_i oprio$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *set-task-ioprio-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(set\text{-}task\text{-}ioprio\ s\ pid\ p\ i)$
and *p6*: $t' = fst(set\text{-}task\text{-}ioprio\ t\ pid\ p\ i)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 set-task-ioprio-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 set-task-ioprio-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *set-task-ioprio-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = TskEvt$ ((*Event-set-task-ioprio pid p i*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' =$ *fst*(*set-task-ioprio s pid p i*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*set-task-ioprio t pid p i*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 set-task-ioprio-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *set-task-ioprio-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-set-task-ioprio pid p i*)))
using *dynamic-weakly-step-consistent-e-def set-task-ioprio-wsc-e*
by *blast*

29.34.9 proving "get$_{t}ask_{i}oprio$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *get-task-ioprio-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*get-task-ioprio s pid p*)
and *p6*: $t' =$ *fst*(*get-task-ioprio t pid p*)
shows $s' \sim d \sim t'$
proof −

{
have *a0* : $s = s'$
using *p5 get-task-ioprio-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 get-task-ioprio-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *get-task-ioprio-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e =$ *TskEvt* ((*Event-get-task-ioprio pid p*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' =$ *fst*(*get-task-ioprio s pid p*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*get-task-ioprio t pid p*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 get-task-ioprio-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *get-task-ioprio-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-get-task-ioprio pid p*)))
using *dynamic-weakly-step-consistent-e-def get-task-ioprio-wsc-e*

by *blast*

29.34.10 proving "check$_p$rlimit$_p$ermission"satisfyingthe"weaklystepconsistent"property

lemma *check-prlimit-permission-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(check\text{-}prlimit\text{-}permission\ s\ pid\ p\ i)$
and *p6*: $t' = fst(check\text{-}prlimit\text{-}permission\ t\ pid\ p\ i)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 check-prlimit-permission-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 check-prlimit-permission-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *check-prlimit-permission-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *TskEvt* ((*Event-check-prlimit-permission pid p i*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' = fst(check\text{-}prlimit\text{-}permission\ s\ pid\ p\ i)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(check\text{-}prlimit\text{-}permission\ t\ pid\ p\ i)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*

using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 check-prlimit-permission-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *check-prlimit-permission-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-check-prlimit-permission pid p i*)))
using *dynamic-weakly-step-consistent-e-def check-prlimit-permission-wsc-e*
by *blast*

29.34.11 proving "do$_{p}rlimit" satisfying the" weakly step consistent" property$

lemma *do-prlimit-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(do\text{-}prlimit\ s\ pid\ p\ i)$
and *p6*: $t' = fst(do\text{-}prlimit\ t\ pid\ p\ i)$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 do-prlimit-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 do-prlimit-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-prlimit-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *TskEvt* ((*Event-do-prlimit pid p i*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$

and *p6*: $s' = $ *exec-event s e*
and *p7*: $t' = $ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' = $ *fst*(*do-prlimit s pid p i*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = $ *fst*(*do-prlimit t pid p i*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-prlimit-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-prlimit-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-do-prlimit pid p i*)))
using *dynamic-weakly-step-consistent-e-def do-prlimit-wsc-e*
by *blast*

29.34.12 proving "task$_{s}etscheduler" satisfyingthe "weaklystepconsistent" property$

lemma *task-setscheduler-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = $ *fst*(*task-setscheduler s pid p*)
and *p6*: $t' = $ *fst*(*task-setscheduler t pid p*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 task-setscheduler-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 task-setscheduler-def*
by (*smt fst-conv*)

have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-setscheduler-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = TskEvt$ ((*Event-task-setscheduler pid p*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' =$ *fst*(*task-setscheduler s pid p*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*task-setscheduler t pid p*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 task-setscheduler-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-setscheduler-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-task-setscheduler pid p*)))
using *dynamic-weakly-step-consistent-e-def task-setscheduler-wsc-e*
by *blast*

29.34.13 proving "task$_getscheduler" satisfying the" weakly step consistent" property$

lemma *task-getscheduler-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*

and $p2$: $s \sim d \sim t$
and $p3$: pid @ s d
and $p4$: $s \sim pid \sim t$
and $p5$: $s' = fst(task\text{-}getscheduler\ s\ pid\ p)$
and $p6$: $t' = fst(task\text{-}getscheduler\ t\ pid\ p)$
shows $s' \sim d \sim t'$
proof −
{
have $a0$: $s = s'$
using *p5 task-getscheduler-def*
by (*smt fstI*)
have $a1$: $t = t'$
using *p6 task-getscheduler-def*
by (*smt fst-conv*)
have $a2$: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-getscheduler-wsc-e*:
assumes $p0$: *reachable0 s*
and $p1$: *reachable0 t*
and $p2$: $e = TskEvt\ ((Event\text{-}task\text{-}getscheduler\ pid\ p))$
and $p3$: $s \sim d \sim t$
and $p4$: $(the\ (domain\text{-}of\text{-}event\ e))$ @ s d
and $p5$: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and $p6$: $s' = exec\text{-}event\ s\ e$
and $p7$: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have $a0$: $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have $a1$: $s' = fst(task\text{-}getscheduler\ s\ pid\ p)$
using *p2 p6 exec-event-def* **by** *auto*
have $a2$: $t' = fst(task\text{-}getscheduler\ t\ pid\ p)$
using *p2 p7 exec-event-def*
by *auto*
have $a3$: pid @ sd
using *p4 a0*
by *blast*
have $a4$: $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have $a5$: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 task-getscheduler-wsc*
by *blast*

}
then show *?thesis* by *auto*
qed

lemma *task-getscheduler-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-task-getscheduler pid p*)))
using *dynamic-weakly-step-consistent-e-def task-getscheduler-wsc-e*
by *blast*

29.34.14 proving "task$_m ovememory" satisfying the" weakly step consistent" property$

lemma *task-movememory-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(task\text{-}movememory\ s\ pid\ p)$
and *p6*: $t' = fst(task\text{-}movememory\ t\ pid\ p)$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 task-movememory-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 task-movememory-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-movememory-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *TskEvt* ((*Event-task-movememory pid p*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-tsk-event-def*

by *force*
have *a1*: $s' = fst(task\text{-}movememory\ s\ pid\ p)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(task\text{-}movememory\ t\ pid\ p)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid @ sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 task-movememory-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-movememory-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-task-movememory pid p*)))
using *dynamic-weakly-step-consistent-e-def task-movememory-wsc-e*
by *blast*

29.34.15 proving "task$_k ill$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *task-kill-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid @ s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(task\text{-}kill\ s\ pid\ p\ sinfo\ i\ c)$
and *p6*: $t' = fst(task\text{-}kill\ t\ pid\ p\ sinfo\ i\ c)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 task-kill-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 task-kill-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-kill-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = TskEvt\ ((Event\text{-}task\text{-}kill\ pid\ p\ sinfo\ i\ c))$
and *p3*: $s \sim d \sim t$
and *p4*: $(the\ (domain\text{-}of\text{-}event\ e))$ @ $s\ d$
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p2 domain-of-event-def getpid-from-tsk-event-def*
by *force*
have *a1*: $s' = fst(task\text{-}kill\ s\ pid\ p\ sinfo\ i\ c)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(task\text{-}kill\ t\ pid\ p\ sinfo\ i\ c)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 task-kill-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-kill-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-task-kill pid p sinfo i c*)))
using *dynamic-weakly-step-consistent-e-def task-kill-wsc-e*
by *blast*

29.34.16 proving "task$_{prctl}$" satisfyingthe" weaklystepconsistent" property

lemma *task-prctl-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(task\text{-}prctl\ s\ pid\ op\ arg2\ arg3\ arg4\ arg5)$
and *p6*: $t' = fst(task\text{-}prctl\ t\ pid\ op\ arg2\ arg3\ arg4\ arg5)$
shows $s' \sim d \sim t'$
proof –

{
  have *a0* : $s = s'$
    using *p5 task-prctl-def*
    by (*smt fstI*)
  have *a1* : $t = t'$
    using *p6 task-prctl-def*
    by (*smt fst-conv*)
  have *a2*: $s' \sim d \sim t'$
    using *a0 a1 p2*
    by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *task-prctl-wsc-e*:
  assumes *p0*: *reachable0 s*
    and *p1*: *reachable0 t*
    and *p2*: $e = $ *TskEvt* ((*Event-task-prctl pid op arg2 arg3 arg4 arg5*))
    and *p3*: $s \sim d \sim t$
    and *p4*: (*the* (*domain-of-event e*)) @ *s d*
    and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
    and *p6*: $s' = $ *exec-event s e*
    and *p7*: $t' = $ *exec-event t e*
  shows $s' \sim d \sim t'$
  proof −
    {
    have *a0* : (*the* (*domain-of-event e*)) = *pid*
      using *p2 domain-of-event-def getpid-from-tsk-event-def*
      by *force*
    have *a1*: $s' = $ *fst*(*task-prctl s pid op arg2 arg3 arg4 arg5*)
      using *p2 p6 exec-event-def* **by** *auto*
    have *a2*: $t' = $ *fst*(*task-prctl t pid op arg2 arg3 arg4 arg5*)
      using *p2 p7 exec-event-def*
      by *auto*
    have *a3*: *pid* @ *sd*
      using *p4 a0*
      by *blast*
    have *a4* : $s \sim$ *pid* $\sim t$ **using** *p5 a0*
      by *blast*
    have *a5*: $s' \sim d \sim t'$
      using *a1 a2 a3 a4 p0 p1 p3 p5 p4 task-prctl-wsc*
      by *blast*
    }
  then show *?thesis* **by** *auto*
qed

lemma *task-prctl-dwsc-e*: *dynamic-weakly-step-consistent-e* (*TskEvt* ((*Event-task-prctl pid op arg2 arg3 arg4 arg5*)))
  using *dynamic-weakly-step-consistent-e-def task-prctl-wsc-e*

by *blast*

29.35 smack ptrace hooks weakly step consistent

29.35.1 proving "ptrace$_{m}ay_{a}ccess$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *ptrace-may-access-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(ptrace\text{-}may\text{-}access\ s\ pid\ p\ m)$
and *p6*: $t' = fst(ptrace\text{-}may\text{-}access\ t\ pid\ p\ m)$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 ptrace-may-access-def*
by *simp*
have *a1* : $t = t'$
using *p6 ptrace-may-access-def*
by *simp*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2* **by** *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ptrace-may-access-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e =$ *PtraceEvt* (*Event-ptrace-access-check pid p m*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-ptrace-event-def*
by *force*
have *a1*: $s' = fst(ptrace\text{-}may\text{-}access\ s\ pid\ p\ m)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(ptrace\text{-}may\text{-}access\ t\ pid\ p\ m)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*

using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 ptrace-may-access-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed
lemma *ptrace-may-access-dwsc-e*: *dynamic-weakly-step-consistent-e* (*PtraceEvt* (*Event-ptrace-access-check pid p m*))
using *dynamic-weakly-step-consistent-e-def ptrace-may-access-wsc-e* **by** *blast*

29.35.2 proving "ptrace$_traceme$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *ptrace-traceme-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(ptrace\text{-}traceme\ s\ pid\ )$
and *p6*: $t' = fst(ptrace\text{-}traceme\ t\ pid\ )$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 ptrace-traceme-def*
by *simp*
have *a1* : $t = t'$
using *p6 ptrace-traceme-def*
by *simp*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2* **by** *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ptrace-traceme-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *PtraceEvt* (*Event-ptrace-traceme pid*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$

proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-ptrace-event-def*
by *force*
have *a1*: $s' = fst(ptrace\text{-}traceme\ s\ pid\ )$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(ptrace\text{-}traceme\ t\ pid\ )$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 ptrace-traceme-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed
lemma *ptrace-traceme-dwsc-e*: *dynamic-weakly-step-consistent-e* (*PtraceEvt* (*Event-ptrace-traceme pid*))
using *dynamic-weakly-step-consistent-e-def ptrace-traceme-wsc-e* **by** *blast*

29.35.3 proving "check$_{s}yslog_{p}ermissions$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *check-syslog-permissions-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst($ *check-syslog-permissions s pid tp*)
and *p6*: $t' = fst($ *check-syslog-permissions t pid tp*)
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 check-syslog-permissions-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 check-syslog-permissions-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*

qed

lemma *check-syslog-permissions-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SysEvt*(*Event-smack-syslog pid tp*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sys-event-def*
by *force*
have *a1*: s' = *fst*(*check-syslog-permissions s pid tp*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*check-syslog-permissions t pid tp*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 check-syslog-permissions-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *check-syslog-permissions-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SysEvt*(
Event-smack-syslog pid t))
using *dynamic-weakly-step-consistent-e-def check-syslog-permissions-wsc-e*
by *blast*

29.35.4 proving "prepare$_b inprm$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *prepare-binprm-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: s' = *fst*(*prepare-binprm s pid bprm*)
and *p6*: t' = *fst*(*prepare-binprm t pid bprm*)

shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 prepare-binprm-def*
proof –
have $s = s' \vee$ *resultValue s* (*security-bprm-set-creds s bprm*) = *0*
using *p5 prepare-binprm-def* **by** *force*
then show *?thesis*
using *p5 prepare-binprm-def* **by** *fastforce*
qed
have *a1* : $t = t'$
using *p6 prepare-binprm-def*
by (*smt fstI*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *prepare-binprm-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SysEvt*(*Event-prepare-binprm pid bprm*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-sys-event-def*
by *force*
have *a1*: s' = *fst*(*prepare-binprm s pid bprm*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*prepare-binprm t pid bprm*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 prepare-binprm-wsc*
by *blast*

    }
  then show *?thesis* **by** *auto*
qed

lemma *prepare-binprm-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SysEvt*(*Event-prepare-binprm pid bprm*))
  using *dynamic-weakly-step-consistent-e-def prepare-binprm-wsc-e*
  by *blast*

29.36 smack ipc hooks weakly step consistent

29.36.1 proving "ipcperms" satisfying the "weakly step consistent" property

lemma *ipcperms-wsc*:
 assumes *p0*: *reachable0 s*
   and *p1*: *reachable0 t*
   and *p2*: $s \sim d \sim t$
   and *p3*: *pid* @ *s d*
   and *p4*: $s \sim pid \sim t$
   and *p5*: $s' = fst(ipcperms\ s\ pid\ ipcp\ flg)$
   and *p6*: $t' = fst(ipcperms\ t\ pid\ ipcp\ flg)$
  shows $s' \sim d \sim t'$
   proof −
  {
   have *a0* : $s = s'$
     using *p5 ipcperms-def*
     by *simp*
   have *a1* : $t = t'$
     using *p6 ipcperms-def*
      by (*smt fstI*)
   have *a2*: $s' \sim d \sim t'$
     using *a0 a1 p2*
     by *blast*
  }
  then show *?thesis* **by** *auto*
qed

lemma *ipcperms-wsc-e*:
  assumes *p0*: *reachable0 s*
   and *p1*: *reachable0 t*
   and *p2*: *e* = *KIpcEvt*((*Event-ipc-permission pid ipcp flg*))
   and *p3*: $s \sim d \sim t$
   and *p4*: (*the* (*domain-of-event e*)) @ *s d*
   and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
   and *p6*: s' = *exec-event s e*
   and *p7*: t' = *exec-event t e*
  shows $s' \sim d \sim t'$
  proof −
   {

have *a0* : (*the* (*domain-of-event e*)) = *pid*
  using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
  by *force*
have *a1*: $s' = fst(ipcperms\ s\ pid\ ipcp\ flg)$
  using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(ipcperms\ t\ pid\ ipcp\ flg)$
  using *p2 p7 exec-event-def*
  by *auto*
have *a3*: *pid* @ *sd*
  using *p4 a0*
  by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
  by *blast*
have *a5*: $s' \sim d \sim t'$
  using *a1 a2 a3 a4 p0 p1 p3 p5 p4 ipcperms-wsc*
  by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ipcperms-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-ipc-permission pid ipcp flg*)))
using *dynamic-weakly-step-consistent-e-def ipcperms-wsc-e*
by *blast*

29.36.2 proving "audit$_{ipc_obj}$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *audit-ipc-obj-wsc*:
assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $s \sim d \sim t$
  and *p3*: *pid* @ *s d*
  and *p4*: $s \sim pid \sim t$
  and *p5*: $s' = fst(audit\text{-}ipc\text{-}obj\ s\ pid\ ipcp)$
  and *p6*: $t' = fst(audit\text{-}ipc\text{-}obj\ t\ pid\ ipcp)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
  using *p5 audit-ipc-obj-def*
  by *simp*
have *a1* : $t = t'$
  using *p6 audit-ipc-obj-def*
  by *force*
have *a2*: $s' \sim d \sim t'$
  using *a0 a1 p2*
  by *blast*
}
then show *?thesis* **by** *auto*

qed

lemma *audit-ipc-obj-wsc-e*:
  assumes *p0*: *reachable0 s*
    and *p1*: *reachable0 t*
    and *p2*: $e = KIpcEvt((Event\text{-}ipc\text{-}getsecid\ pid\ ipcp))$
    and *p3*: $s \sim d \sim t$
    and *p4*: $(the\ (domain\text{-}of\text{-}event\ e))$ @ $s\ d$
    and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
    and *p6*: $s' = exec\text{-}event\ s\ e$
    and *p7*: $t' = exec\text{-}event\ t\ e$
  shows $s' \sim d \sim t'$
  proof −
    {
    have *a0* : $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
      using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
      by *force*
    have *a1*: $s' = fst(audit\text{-}ipc\text{-}obj\ s\ pid\ ipcp)$
      using *p2 p6 exec-event-def* **by** *auto*
    have *a2*: $t' = fst(audit\text{-}ipc\text{-}obj\ t\ pid\ ipcp)$
      using *p2 p7 exec-event-def*
      by *auto*
    have *a3*: *pid* @ *sd*
      using *p4 a0*
      by *blast*
    have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
      by *blast*
    have *a5*: $s' \sim d \sim t'$
       using *a1 a2 a3 a4 p0 p1 p3 p5 p4 audit-ipc-obj-wsc*
       by *blast*
     }
  then show *?thesis* **by** *auto*
qed

lemma *audit-ipc-obj-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-ipc-getsecid pid ipcp*)))
  using *dynamic-weakly-step-consistent-e-def audit-ipc-obj-wsc-e*
  by *blast*

29.36.3 proving "ksys$_msgget$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *ksys-msgget-wsc*:
 assumes *p0*: *reachable0 s*
    and *p1*: *reachable0 t*
    and *p2*: $s \sim d \sim t$
    and *p3*: *pid* @ *s d*
    and *p4*: $s \sim pid \sim t$
    and *p5*: $s' = fst(ksys\text{-}msgget\ s\ pid\ msq\ msqflg)$
    and *p6*: $t' = fst(ksys\text{-}msgget\ t\ pid\ msq\ msqflg)$

shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 ksys-msgget-def*
by *simp*
have *a1* : $t = t'$
using *p6 ksys-msgget-def*
by (*smt fstI*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ksys-msgget-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *KIpcEvt*((*Event-msg-queue-associate pid msq msqflg*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
by *force*
have *a1*: s' = *fst*(*ksys-msgget s pid msq msqflg*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*ksys-msgget t pid msq msqflg*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 ksys-msgget-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ksys-msgget-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-msg-queue-associate*

pid msq msqflg)))
  using *dynamic-weakly-step-consistent-e-def ksys-msgget-wsc-e*
  by *blast*

29.36.4 proving "msg$_{q}ueue_{m}sgctl$" $satisfying the$" $weakly step consistent$" $property$

lemma *msg-queue-msgctl-wsc*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $s \sim d \sim t$
  and *p3*: *pid* @ *s d*
  and *p4*: $s \sim pid \sim t$
  and *p5*: $s' = fst(msg\text{-}queue\text{-}msgctl\ s\ pid\ msq\ cmd)$
  and *p6*: $t' = fst(msg\text{-}queue\text{-}msgctl\ t\ pid\ msq\ cmd)$
 shows $s' \sim d \sim t'$
 proof −
 {
  have *a0* : $s = s'$
   using *p5 msg-queue-msgctl-def*
   by *simp*
  have *a1* : $t = t'$
   using *p6 msg-queue-msgctl-def*
   by *auto*
  have *a2*: $s' \sim d \sim t'$
   using *a0 a1 p2*
   by *blast*
 }
 then show *?thesis* **by** *auto*
qed

lemma *msg-queue-msgctl-wsc-e*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $e =$ *KIpcEvt*((*Event-msg-queue-msgctl pid msq cmd*))
  and *p3*: $s \sim d \sim t$
  and *p4*: (*the* (*domain-of-event e*)) @ *s d*
  and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
  and *p6*: $s' =$ *exec-event s e*
  and *p7*: $t' =$ *exec-event t e*
 shows $s' \sim d \sim t'$
 proof −
  {
  have *a0* : (*the* (*domain-of-event e*)) = *pid*
   using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
   by *force*
  have *a1*: $s' = fst(msg\text{-}queue\text{-}msgctl\ s\ pid\ msq\ cmd)$
   using *p2 p6 exec-event-def* **by** *auto*
  have *a2*: $t' = fst(msg\text{-}queue\text{-}msgctl\ t\ pid\ msq\ cmd)$
   using *p2 p7 exec-event-def*

by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 msg-queue-msgctl-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *msg-queue-msgctl-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-msg-queue-msgctl pid msq cmd*)))
using *dynamic-weakly-step-consistent-e-def msg-queue-msgctl-wsc-e*
by *blast*

29.36.5 proving "do$_m sgsnd$" $satisfying the$ "$weakly step consistent$" $property$

lemma *do-msgsnd-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(do\text{-}msgsnd\ s\ pid\ msq\ msg\ msgflg)$
and *p6*: $t' = fst(do\text{-}msgsnd\ t\ pid\ msq\ msg\ msgflg)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 do-msgsnd-def*
by (*smt do-msgsnd-def fst-conv p5*)
have *a1* : $t = t'$
using *p6 do-msgsnd-def*
by (*smt fstI*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-msgsnd-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *KIpcEvt*((*Event-msg-queue-msgsnd pid msq msg msgflg*))
and *p3*: $s \sim d \sim t$

and $p4$: (*the* (*domain-of-event e*)) @ *s d*
and $p5$: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and $p6$: $s' =$ *exec-event s e*
and $p7$: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
by *force*
have *a1*: $s' =$ *fst*(*do-msgsnd s pid msq msg msgflg*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*do-msgsnd t pid msq msg msgflg*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-msgsnd-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-msgsnd-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-msg-queue-msgsnd pid msq msg msgflg*)))
using *dynamic-weakly-step-consistent-e-def do-msgsnd-wsc-e*
by *blast*

29.36.6 proving "msg$_q ueue_m sgrcv$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *msg-queue-msgrcv-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*msg-queue-msgrcv s pid isp msq p long msqflg*)
and *p6*: $t' =$ *fst*(*msg-queue-msgrcv t pid isp msq p long msqflg*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 msg-queue-msgrcv-def*
by *simp*
have *a1* : $t = t'$

using *p6 msg-queue-msgrcv-def*
by *auto*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* by *auto*
qed

lemma *msg-queue-msgrcv-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = KIpcEvt(\ (Event\text{-}msg\text{-}queue\text{-}msgrcv\ \ pid\ isp\ msq\ p\ long\ msqflg))$
and *p3*: $s \sim d \sim t$
and *p4*: $(the\ (domain\text{-}of\text{-}event\ e))\ @\ s\ d$
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
by *force*
have *a1*: $s' = fst(msg\text{-}queue\text{-}msgrcv\ s\ pid\ isp\ msq\ p\ long\ msqflg)$
using *p2 p6 exec-event-def* by *auto*
have *a2*: $t' = fst(msg\text{-}queue\text{-}msgrcv\ t\ pid\ isp\ msq\ p\ long\ msqflg)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: $pid\ @\ sd$
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ using *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 msg-queue-msgrcv-wsc*
by *blast*
}
then show *?thesis* by *auto*
qed

lemma *msg-queue-msgrcv-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-msg-queue-msgrcv pid isp msq p long msqflg*)))
using *dynamic-weakly-step-consistent-e-def msg-queue-msgrcv-wsc-e*
by *blast*

29.36.7 proving "ksys$_shmget$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *ksys-shmget-wsc*:

assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(ksys\text{-}shmget\ s\ pid\ shm\ shmflg)$
and *p6*: $t' = fst(ksys\text{-}shmget\ t\ pid\ shm\ shmflg)$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 ksys-shmget-def*
by *simp*
have *a1* : $t = t'$
using *p6 ksys-shmget-def*
by *auto*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ksys-shmget-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = KIpcEvt(\ (Event\text{-}shm\text{-}associate\ pid\ shm\ shmflg))$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
by *force*
have *a1*: $s' = fst(ksys\text{-}shmget\ s\ pid\ shm\ shmflg)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(ksys\text{-}shmget\ t\ pid\ shm\ shmflg)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$

using *a1 a2 a3 a4 p0 p1 p3 p4 p5 ksys-shmget-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ksys-shmget-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-shm-associate pid shm shmflg*)))
using *dynamic-weakly-step-consistent-e-def ksys-shmget-wsc-e*
by *blast*

29.36.8 proving "shm$_m$*sgctl*"*satisfyingthe*"*weaklystepconsistent*"*property*

lemma *shm-msgctl-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(shm\text{-}msgctl\ s\ pid\ shm\ cmd)$
and *p6*: $t' = fst(shm\text{-}msgctl\ t\ pid\ shm\ cmd)$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 shm-msgctl-def*
by *simp*
have *a1* : $t = t'$
using *p6 shm-msgctl-def*
by *auto*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *shm-msgctl-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = KIpcEvt(\ (Event\text{-}shm\text{-}shmctl\ pid\ shm\ cmd\ ))$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof –
{

have *a0* : (*the* (*domain-of-event e*)) = *pid*
  using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
  by *force*
have *a1*: *s'* = *fst*(*shm-msgctl s pid shm cmd*)
  using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t'* = *fst*(*shm-msgctl t pid shm cmd*)
  using *p2 p7 exec-event-def*
  by *auto*
have *a3*: *pid* @ *sd*
  using *p4 a0*
  by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
  by *blast*
have *a5*: *s'* ∼ *d* ∼ *t'*
  using *a1 a2 a3 a4 p0 p1 p3 shm-msgctl-wsc*
  by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *shm-msgctl-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-shm-shmctl pid shm cmd*)))
  using *dynamic-weakly-step-consistent-e-def shm-msgctl-wsc-e*
  by *blast*

29.36.9 proving "do$_{s}hmat$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *do-shmat-wsc*:
assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: *s* ∼ *d* ∼ *t*
  and *p3*: *pid* @ *s d*
  and *p4*: *s* ∼ *pid* ∼ *t*
  and *p5*: *s'* = *fst*(*do-shmat s pid shp shmaddr shmflg*)
  and *p6*: *t'* = *fst*(*do-shmat t pid shp shmaddr shmflg*)
shows *s'* ∼ *d* ∼ *t'*
proof −
{
have *a0* : *s* = *s'*
  using *p5*
  apply(*simp add*: *do-shmat-def*)
  apply *auto*[*1*]
  apply (*smt fst-conv*)
  by (*smt eq-fst-iff*)
have *a1* : *t* = *t'*
  using *p6*
  apply(*simp add*: *do-shmat-def*)
  apply *auto*[*1*]
  apply (*smt fst-conv*)

        by (*smt eq-fst-iff*)
    have *a2*: $s' \sim d \sim t'$
      using *a0 a1 p2*
      by *blast*
  }
  then show *?thesis* **by** *auto*
qed

lemma *do-shmat-wsc-e*:
  assumes *p0*: *reachable0 s*
    and *p1*: *reachable0 t*
    and *p2*: e = *KIpcEvt*((*Event-shm-shmat pid shp shmaddr shmflg*))
    and *p3*: $s \sim d \sim t$
    and *p4*: (*the* (*domain-of-event e*)) @ *s d*
    and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
    and *p6*: s' = *exec-event s e*
    and *p7*: t' = *exec-event t e*
  shows $s' \sim d \sim t'$
  proof −
    {
    have *a0* : (*the* (*domain-of-event e*)) = *pid*
      using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
      by *force*
    have *a1*: s' = *fst*(*do-shmat s pid shp shmaddr shmflg*)
      using *p2 p6 exec-event-def* **by** *auto*
    have *a2*: t' = *fst*(*do-shmat t pid shp shmaddr shmflg*)
      using *p2 p7 exec-event-def*
      by *auto*
    have *a3*: *pid* @ *sd*
      using *p4 a0*
      by *blast*
    have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
      by *blast*
    have *a5*: $s' \sim d \sim t'$
       using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-shmat-wsc*
       by *blast*
     }
  then show *?thesis* **by** *auto*
qed

lemma *do-shmat-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-shm-shmat pid shp shmaddr shmflg*)))
  using *dynamic-weakly-step-consistent-e-def do-shmat-wsc-e*
  by *blast*

29.36.10 proving "ksys$_s$emget"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *ksys-semget-wsc*:
 assumes *p0*: *reachable0 s*

```
    and   p1: reachable0 t
    and   p2: s ∼ d ∼ t
    and   p3: pid @ s d
    and   p4: s ∼ pid ∼ t
    and   p5: s' = fst(ksys-semget s pid sem semflg)
    and   p6: t' = fst(ksys-semget t pid sem semflg)
  shows   s' ∼ d ∼ t'
   proof −
  {
    have a0 : s = s'
      using p5 ksys-semget-def
      by simp
    have a1 : t = t'
      using p6 ksys-semget-def
       by auto
    have a2: s' ∼ d ∼ t'
      using a0 a1 p2
      by blast
  }
  then show ?thesis by auto
qed

lemma ksys-semget-wsc-e:
  assumes p0: reachable0 s
    and   p1: reachable0 t
    and   p2: e =  KIpcEvt( (Event-sem-associate pid sem semflg ))
    and   p3: s ∼ d ∼ t
    and   p4: (the (domain-of-event e)) @ s d
    and   p5: s ∼ (the (domain-of-event e)) ∼ t
    and   p6: s' = exec-event s e
    and   p7: t' = exec-event t e
  shows   s' ∼ d ∼ t'
  proof −
   {
    have a0 :  (the (domain-of-event e)) = pid
      using p2 domain-of-event-def getpid-from-kern-ipc-event-def
      by force
    have a1: s' =  fst(ksys-semget s pid sem semflg)
      using p2 p6 exec-event-def by auto
    have a2: t' =  fst(ksys-semget t pid sem semflg)
      using p2 p7 exec-event-def
      by auto
    have a3: pid @ sd
      using p4 a0
      by blast
    have a4 : s ∼ pid ∼ t using p5 a0
      by blast
    have a5: s' ∼ d ∼ t'
       using a1 a2 a3 a4 p0 p1 p3 p4 p5 ksys-semget-wsc
```

by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ksys-semget-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-sem-associate pid sem semflg*)))
using *dynamic-weakly-step-consistent-e-def ksys-semget-wsc-e*
by *blast*

29.36.11 proving "sem$_m sgctl$" $satisfying the$" $weakly step consistent$" $property$

lemma *sem-msgctl-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(sem\text{-}msgctl\ s\ pid\ sem\ cmd)$
and *p6*: $t' = fst(sem\text{-}msgctl\ t\ pid\ sem\ cmd)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 sem-msgctl-def*
by *simp*
have *a1* : $t = t'$
using *p6 sem-msgctl-def*
by *auto*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sem-msgctl-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *KIpcEvt*((*Event-sem-semctl pid sem cmd*))
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*

using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
by *force*
have *a1*: $s' =$ *fst*(*sem-msgctl s pid sem cmd*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*sem-msgctl t pid sem cmd*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p4 p5 sem-msgctl-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sem-msgctl-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-sem-semctl pid sem cmd*)))
using *dynamic-weakly-step-consistent-e-def sem-msgctl-wsc-e*
by *blast*

29.36.12 proving "do$_s$*emtimedop*" *satisfyingthe*" *weaklystepconsistent*" *property*

lemma *do-semtimedop-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*do-semtimedop s pid sma sops nsops alter'*)
and *p6*: $t' =$ *fst*(*do-semtimedop t pid sma sops nsops alter'*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 do-semtimedop-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 do-semtimedop-def*
by (*smt fstI*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-semtimedop-wsc-e*:
  assumes *p0*: *reachable0 s*
    and *p1*: *reachable0 t*
    and *p2*: *e* = *KIpcEvt*((*Event-sem-semop pid sma sops nsops alter'*))
    and *p3*: *s* ∼ *d* ∼ *t*
    and *p4*: (*the* (*domain-of-event e*)) @ *s d*
    and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
    and *p6*: *s'* = *exec-event s e*
    and *p7*: *t'* = *exec-event t e*
  shows *s'* ∼ *d* ∼ *t'*
  proof −
    {
    have *a0* : (*the* (*domain-of-event e*)) = *pid*
      using *p2 domain-of-event-def getpid-from-kern-ipc-event-def*
      by *force*
    have *a1*: *s'* = *fst*(*do-semtimedop s pid sma sops nsops alter'*)
      using *p2 p6 exec-event-def* **by** *auto*
    have *a2*: *t'* = *fst*(*do-semtimedop t pid sma sops nsops alter'*)
      using *p2 p7 exec-event-def*
      by *auto*
    have *a3*: *pid* @ *sd*
      using *p4 a0*
      by *blast*
    have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
      by *blast*
    have *a5*: *s'* ∼ *d* ∼ *t'*
       using *a1 a2 a3 a4 p0 p1 p3 p4 p5 do-semtimedop-wsc*
       by *blast*
     }
  then show *?thesis* **by** *auto*
qed

lemma *do-semtimedop-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KIpcEvt*((*Event-sem-semop pid sma sops nsops alter'*)))
  using *dynamic-weakly-step-consistent-e-def do-semtimedop-wsc-e*
  by *blast*

29.37 smack file hooks weakly step consistent

29.37.1 proving "do$_i$*octl*"*satisfyingthe*"*weaklystepconsistent*"*property*

lemma *do-ioctl-wsc*:
 assumes *p0*: *reachable0 s*
    and *p1*: *reachable0 t*
    and *p2*: *s* ∼ *d* ∼ *t*
    and *p3*: *pid* @ *s d*
    and *p4*: *s* ∼ *pid* ∼ *t*
    and *p5*: *s'* = *fst*(*do-ioctl s pid file cmd arg*)
    and *p6*: *t'* = *fst*(*do-ioctl t pid file cmd arg*)

shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 do-ioctl-def do-ioctl-detstate fst-conv funcState-def security-file-ioctl-notchgstate*
by *smt*
have *a1* : $t = t'$
using *p6 do-ioctl-def do-ioctl-detstate fst-conv funcState-def security-file-ioctl-notchgstate*
by *smt*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-ioctl-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = FileEvt($ *Event-do-ioctl pid file cmd arg* $)$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: $s' =$ *fst*(*do-ioctl s pid file cmd arg*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*do-ioctl t pid file cmd arg*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *s d*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-ioctl-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-ioctl-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-do-ioctl* *pid file cmd arg*))
  using *dynamic-weakly-step-consistent-e-def do-ioctl-wsc-e*
  by *blast*

29.37.2 proving "syscall$_{i}octl" satisfyingthe" weaklystepconsistent" property$

lemma *syscall-ioctl-wsc*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $s \sim d \sim t$
  and *p3*: *pid* @ *s d*
  and *p4*: $s \sim pid \sim t$
  and *p5*: $s' = fst(syscall\text{-}ioctl\ s\ pid\ fd\ cmd\ arg)$
  and *p6*: $t' = fst(syscall\text{-}ioctl\ t\ pid\ fd\ cmd\ arg)$
 shows $s' \sim d \sim t'$
  proof −
 {
  have *a0* : $s = s'$
   using *p5 syscall-ioctl-def*
   by (*smt fstI*)
  have *a1* : $t = t'$
   using *p6 syscall-ioctl-def*
   by (*smt fst-conv*)
  have *a2*: $s' \sim d \sim t'$
   using *a0 a1 p2*
   by *blast*
 }
 then show *?thesis* **by** *auto*
qed

lemma *syscall-ioctl-wsc-e*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: *e* = *FileEvt*(*Event-syscall-ioctl pid fd cmd arg*)
  and *p3*: $s \sim d \sim t$
  and *p4*: (*the* (*domain-of-event e*)) @ *s d*
  and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
  and *p6*: $s' = exec\text{-}event\ s\ e$
  and *p7*: $t' = exec\text{-}event\ t\ e$
 shows $s' \sim d \sim t'$
 proof −
  {
  have *a0* : (*the* (*domain-of-event e*)) = *pid*
   using *p2 domain-of-event-def getpid-from-file-event-def*
   by *force*
  have *a1*: $s' = fst(syscall\text{-}ioctl\ s\ pid\ fd\ cmd\ arg)$
   using *p2 p6 exec-event-def* **by** *auto*

have *a2*: $t' = fst(syscall\text{-}ioctl\ t\ pid\ fd\ cmd\ arg)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid @ s d*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 syscall-ioctl-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *syscall-ioctl-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-syscall-ioctl pid fd cmd arg*))
using *dynamic-weakly-step-consistent-e-def syscall-ioctl-wsc-e*
by *blast*

29.37.3 proving "ksys$_ioctl$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *ksys-ioctl-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid @ s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(ksys\text{-}ioctl\ s\ pid\ fd\ cmd\ arg)$
and *p6*: $t' = fst(ksys\text{-}ioctl\ t\ pid\ fd\ cmd\ arg)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 ksys-ioctl-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 ksys-ioctl-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ksys-ioctl-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*

and *p2*: *e* = *FileEvt*(*Event-ksys-ioctl pid fd cmd arg*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s'* = *exec-event s e*
and *p7*: *t'* = *exec-event t e*
shows *s'* ∼ *d* ∼ *t'*
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: *s'* = *fst*(*ksys-ioctl s pid fd cmd arg*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t'* = *fst*(*ksys-ioctl t pid fd cmd arg*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s'* ∼ *d* ∼ *t'*
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 ksys-ioctl-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *ksys-ioctl-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-ksys-ioctl pid fd cmd arg*))
using *dynamic-weakly-step-consistent-e-def ksys-ioctl-wsc-e*
by *blast*

29.37.4 proving "vm$_m$$map_pgoff$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *vm-mmap-pgoff-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s'* = *fst*(*vm-mmap-pgoff s pid file addr len' prot flag pgoff*)
and *p6*: *t'* = *fst*(*vm-mmap-pgoff t pid file addr len' prot flag pgoff*)
shows *s'* ∼ *d* ∼ *t'*
proof –
{
have *a0* : *s* = *s'*
using *p5 vm-mmap-pgoff-def*

by (*smt fstI*)
have *a1* : $t = t'$
using *p6 vm-mmap-pgoff-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vm-mmap-pgoff-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: e = *FileEvt*(*Event-vm-mmap-pgoff pid file addr len′ prot flag pgoff*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: s' = *fst*(*vm-mmap-pgoff s pid file addr len′ prot flag pgoff*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*vm-mmap-pgoff t pid file addr len′ prot flag pgoff*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 vm-mmap-pgoff-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vm-mmap-pgoff-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-vm-mmap-pgoff pid file addr len′ prot flag pgoff*))
using *dynamic-weakly-step-consistent-e-def vm-mmap-pgoff-wsc-e*
by *blast*

29.37.5 proving "do$_{sys_v}m86$" $satisfying the$" $weakly step consistent$" $property$

lemma *do-sys-vm86-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(do\text{-}sys\text{-}vm86\ s\ pid)$
and *p6*: $t' = fst(do\text{-}sys\text{-}vm86\ t\ pid)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 do-sys-vm86-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 do-sys-vm86-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-sys-vm86-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = FileEvt(\ Event\text{-}do\text{-}sys\text{-}vm86\ pid\ )$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: $s' = fst(do\text{-}sys\text{-}vm86\ s\ pid)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(do\text{-}sys\text{-}vm86\ t\ pid)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*

by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-sys-vm86-wsc*
by *blast*
}
then show *?thesis* by *auto*
qed

lemma *do-sys-vm86-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-do-sys-vm86 pid*))
using *dynamic-weakly-step-consistent-e-def do-sys-vm86-wsc-e*
by *blast*

29.37.6 proving "get$_u$nmapped$_a$rea" satisfyingthe" weaklystepconsistent" property

lemma *get-unmapped-area-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(get\text{-}unmapped\text{-}area\ s\ pid\ file\ addr)$
and *p6*: $t' = fst(get\text{-}unmapped\text{-}area\ t\ pid\ file\ addr)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 get-unmapped-area-def*
proof −
have $s = s' \vee resultValue\ s\ (security\text{-}mmap\text{-}addr\ s\ addr) = 0$
using *get-unmapped-area-def p5* by *fastforce*
then show *?thesis*
using *get-unmapped-area-def p5* by *force*
qed

have *a1* : $t = t'$
using *p6 get-unmapped-area-def*
proof −
have $t = t' \vee resultValue\ t\ (security\text{-}mmap\text{-}addr\ t\ addr) = 0$
using *get-unmapped-area-def p6* by *force*
then show *?thesis*
using *get-unmapped-area-def p6* by *force*
qed
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* by *auto*
qed

lemma *get-unmapped-area-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *FileEvt*(*Event-get-unmapped-area pid file addr*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s'* = *exec-event s e*
and *p7*: *t'* = *exec-event t e*
shows *s'* ∼ *d* ∼ *t'*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: *s'* = *fst*(*get-unmapped-area s pid file addr*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t'* = *fst*(*get-unmapped-area t pid file addr*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s'* ∼ *d* ∼ *t'*
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 get-unmapped-area-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *get-unmapped-area-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-get-unmapped-area pid file addr*))
using *dynamic-weakly-step-consistent-e-def get-unmapped-area-wsc-e*
by *blast*

29.37.7 proving "validate$_m$$map_request$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *validate-mmap-request-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s'* = *fst*(*validate-mmap-request s pid file addr*)
and *p6*: *t'* = *fst*(*validate-mmap-request t pid file addr*)
shows *s'* ∼ *d* ∼ *t'*

proof −
{
  have *a0* : $s = s'$
    using *p5 validate-mmap-request-def*
    by (*smt fstI*)
  have *a1* : $t = t'$
    using *p6 validate-mmap-request-def*
  proof −
    { **assume** $t \neq t'$
      then have $\neg$ *0* $\leq$ *resultValue t* (*security-mmap-addr t addr*)
        using *p6 validate-mmap-request-def* **by** *force*
      then have *?thesis*
        using *p6 validate-mmap-request-def* **by** *auto* }
    then show *?thesis*
      by *metis*
  qed
  have *a2*: $s' \sim d \sim t'$
    using *a0 a1 p2*
    by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *validate-mmap-request-wsc-e*:
  assumes *p0*: *reachable0 s*
    and *p1*: *reachable0 t*
    and *p2*: *e* = *FileEvt*(*Event-validate-mmap-request pid file addr*)
    and *p3*: $s \sim d \sim t$
    and *p4*: (*the* (*domain-of-event e*)) @ *s d*
    and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
    and *p6*: s' = *exec-event s e*
    and *p7*: t' = *exec-event t e*
  shows $s' \sim d \sim t'$
  proof −
    {
    have *a0* : (*the* (*domain-of-event e*)) = *pid*
      using *p2 domain-of-event-def getpid-from-file-event-def*
      by *force*
    have *a1*: s' = *fst*(*validate-mmap-request s pid file addr*)
      using *p2 p6 exec-event-def* **by** *auto*
    have *a2*: t' = *fst*(*validate-mmap-request t pid file addr*)
      using *p2 p7 exec-event-def*
      by *auto*
    have *a3*: *pid* @ *sd*
      using *p4 a0*
      by *blast*
    have *a4* : $s \sim$ *pid* $\sim t$ **using** *p5 a0*
      by *blast*
    have *a5*: $s' \sim d \sim t'$

using *a1 a2 a3 a4 p0 p1 p3 p5 p4 validate-mmap-request-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *validate-mmap-request-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-validate-mmap-request pid file addr*))
using *dynamic-weakly-step-consistent-e-def validate-mmap-request-wsc-e*
by *blast*

29.37.8 proving "generic$_setlease$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *generic-setlease-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*generic-setlease s pid file arg*)
and *p6*: $t' =$ *fst*(*generic-setlease t pid file arg*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 generic-setlease-def*
by (*smt fst-conv*)
have *a1* : $t = t'$
using *p6 generic-setlease-def*
proof −
{ **assume** $t \neq t'$
then have *resultValue t* (*security-file-lock t file* (*nat arg*)) $\neq 0$
using *generic-setlease-def p6* **by** *force*
then have *?thesis*
by (*simp add*: *generic-setlease-def p6*) }
then show *?thesis*
by *metis*
qed
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *generic-setlease-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *FileEvt*(*Event-generic-setlease pid file arg*)

and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: s' = *fst*(*generic-setlease s pid file arg*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*generic-setlease t pid file arg*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 generic-setlease-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *generic-setlease-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-generic-setlease pid file arg*))
using *dynamic-weakly-step-consistent-e-def generic-setlease-wsc-e*
by *blast*

29.37.9 proving "syscall$_{lock}$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *syscall-lock-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*syscall-lock s pid fd cmd*)
and *p6*: $t' =$ *fst*(*syscall-lock t pid fd cmd*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
by (*simp add*: *p5 syscall-lock-def*)
have *a1* : $t = t'$

using *p6*
by (*simp add*: *syscall-lock-def*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* by *auto*
qed

lemma *syscall-lock-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *FileEvt*(*Event-syscall-lock pid fd cmd*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: s' = *fst*(*syscall-lock s pid fd cmd*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*syscall-lock t pid fd cmd*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 syscall-lock-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *syscall-lock-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-syscall-lock pid fd cmd*))
using *dynamic-weakly-step-consistent-e-def syscall-lock-wsc-e*
by *blast*

29.37.10 proving "do$_lock_file_wait$" satisfying the "weakly step consistent" property

lemma *do-lock-file-wait-wsc*:

assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*do-lock-file-wait s pid file cmd fl*)
and *p6*: $t' =$ *fst*(*do-lock-file-wait t pid file cmd fl*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5*
by (*smt do-lock-file-wait-def fst-conv*)
have *a1* : $t = t'$
using *p6 do-lock-file-wait-def*
proof −
have ∀ *s n f i fa. do-lock-file-wait s n f i fa* =
(*if resultValue s* (*security-file-lock s f* (*nat* (*of-char* (*fl-type fa*)))) = *0*
then (*s*, *0*)
else (*s*, *resultValue s* (*security-file-lock s f* (*nat* (*of-char* (*fl-type fa*))))))
using *do-lock-file-wait-def* **by** *presburger*
then show *?thesis*
by (*metis fstI p6*)
qed
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-lock-file-wait-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *FileEvt*(*Event-do-lock-file-wait pid file cmd fl*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: $s' =$ *fst*(*do-lock-file-wait s pid file cmd fl*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*do-lock-file-wait t pid file cmd fl*)

using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *s d*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s′* ∼ *d* ∼ *t′*
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-lock-file-wait-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-lock-file-wait-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-do-lock-file-wait pid file cmd fl*))
using *dynamic-weakly-step-consistent-e-def do-lock-file-wait-wsc-e*
by *blast*

29.37.11 proving "file$_f$*cntl" satisfyingthe" weaklystepconsistent" property*

lemma *file-fcntl-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s′* = *fst*(*file-fcntl s pid file cmd arg*)
and *p6*: *t′* = *fst*(*file-fcntl t pid file cmd arg*)
shows *s′* ∼ *d* ∼ *t′*
proof −
{
have *a0* : *s* = *s′*
using *p5 file-fcntl-def*
by (*smt fstI*)
have *a1* : *t* = *t′*
using *p6 file-fcntl-def*
proof −
have ∀ *s n f na nb*. *file-fcntl s n f na nb* =
(*if resultValue s* (*security-file-fcntl s f na nb*) = *0*
then (*s*, *0*)
else (*s*, *resultValue s* (*security-file-fcntl s f na nb*)))
using *file-fcntl-def* **by** *presburger*
then have *file-fcntl t pid file cmd arg* = (*t*, *resultValue t* (*security-file-fcntl t file cmd arg*))
by *auto*
then show *?thesis*
using *p6* **by** *auto*
qed

have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *file-fcntl-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = \textit{FileEvt}($ *Event-file-fcntl pid file cmd arg* $)$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: $s' =$ *fst*(*file-fcntl s pid file cmd arg*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*file-fcntl t pid file cmd arg*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim$ *pid* $\sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 file-fcntl-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *file-fcntl-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-file-fcntl pid file cmd arg*))
using *dynamic-weakly-step-consistent-e-def file-fcntl-wsc-e*
by *blast*

29.37.12 proving "file$_{s}end_{s}igiotask$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *file-send-sigiotask-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*

and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*file-send-sigiotask s pid ty fown sig*)
and *p6*: $t' =$ *fst*(*file-send-sigiotask t pid ty fown sig*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 file-send-sigiotask-def*
by *simp*
have *a1* : $t = t'$
using *p6 file-send-sigiotask-def*
by *simp*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *file-send-sigiotask-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *FileEvt*(*Event-file-send-sigiotask pid ty fown sig*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: $s' =$ *fst*(*file-send-sigiotask s pid ty fown sig*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*file-send-sigiotask t pid ty fown sig*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 file-send-sigiotask-wsc*
by *blast*

}
then show *?thesis* **by** *auto*
qed

lemma *file-send-sigiotask-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-file-send-sigiotask pid ty fown sig*))
using *dynamic-weakly-step-consistent-e-def file-send-sigiotask-wsc-e*
by *blast*

29.37.13 proving "file$_receive$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *file-receive-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*file-receive s pid f*)
and *p6*: $t' =$ *fst*(*file-receive t pid f*)
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 file-receive-def*
by *auto*
have *a1* : $t = t'$
using *p6 file-receive-def*
by *simp*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *file-receive-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *FileEvt*(*Event-file-receive pid f*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*

by *force*
have *a1*: $s' = fst(file\text{-}receive\ s\ pid\ f)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(file\text{-}receive\ t\ pid\ f)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid @ sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 file-receive-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *file-receive-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-file-receive pid f*))
using *dynamic-weakly-step-consistent-e-def file-receive-wsc-e*
by *blast*

29.37.14 proving "do$_d entry_o pen$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *do-dentry-open-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid @ s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(do\text{-}dentry\text{-}open\ s\ pid\ f)$
and *p6*: $t' = fst(do\text{-}dentry\text{-}open\ t\ pid\ f)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 do-dentry-open-def* **by** *simp*
have *a1* : $t = t'$
using *p6 do-dentry-open-def* **by** *auto*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-dentry-open-wsc-e*:

assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *FileEvt*(*Event-do-dentry-open pid f*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s'* = *exec-event s e*
and *p7*: *t'* = *exec-event t e*
shows *s'* ∼ *d* ∼ *t'*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-file-event-def*
by *force*
have *a1*: *s'* = *fst*(*do-dentry-open s pid f*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t'* = *fst*(*do-dentry-open t pid f*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s'* ∼ *d* ∼ *t'*
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 do-dentry-open-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *do-dentry-open-dwsc-e*: *dynamic-weakly-step-consistent-e* (*FileEvt*(*Event-do-dentry-open pid f*))
using *dynamic-weakly-step-consistent-e-def do-dentry-open-wsc-e*
by *blast*

29.38 smack key hooks weakly step consistent

29.38.1 proving "key$_{t}ask_{p}ermission" satisfying the "weakly step consistent" property$

lemma *key-task-permission-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s'* = *fst*(*key-task-permission s pid key-ref cred' perm*)
and *p6*: *t'* = *fst*(*key-task-permission t pid key-ref cred' perm*)
shows *s'* ∼ *d* ∼ *t'*
proof −

{
have *a0* : $s = s'$
using *p5 key-task-permission-def*
by *simp*
have *a1* : $t = t'$
using *p6*
using *key-task-permission-def* by *auto*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* by *auto*
qed

lemma *key-task-permission-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = KeyEvt($ *Event-key-permission pid key-ref cred' perm* $)$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-key-evevt-def*
by *force*
have *a1*: $s' =$ *fst*(*key-task-permission s pid key-ref cred' perm*)
using *p2 p6 exec-event-def* by *auto*
have *a2*: $t' =$ *fst*(*key-task-permission t pid key-ref cred' perm*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ using *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 key-task-permission-wsc*
by *blast*
}
then show *?thesis* by *auto*
qed

lemma *key-task-permission-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KeyEvt*(*Event-key-permission pid key-ref cred' perm*))
using *dynamic-weakly-step-consistent-e-def key-task-permission-wsc-e*

by *blast*

29.38.2 proving "keyctl$_{get_security}$" satisfying the "weakly step consistent" property

lemma *keyctl-get-security-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*keyctl-get-security s pid keyid' buffer buflen*)
and *p6*: $t' =$ *fst*(*keyctl-get-security t pid keyid' buffer buflen*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 keyctl-get-security-def*
by *auto*
have *a1* : $t = t'$
using *p6 keyctl-get-security-def*
by *simp*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *keyctl-get-security-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *KeyEvt*(*Event-key-getsecurity pid keyid' buffer buflen*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-key-evevt-def*
by *force*
have *a1*: s' = *fst*(*keyctl-get-security s pid keyid' buffer buflen*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*keyctl-get-security t pid keyid' buffer buflen*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*

using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 keyctl-get-security-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *keyctl-get-security-dwsc-e*: *dynamic-weakly-step-consistent-e* (*KeyEvt*(*Event-key-getsecurity pid keyid' buffer buflen*))
using *dynamic-weakly-step-consistent-e-def keyctl-get-security-wsc-e*
by *blast*

29.39 smack audit hooks weakly step consistent

29.39.1 proving "audit$_{d}ata_{t}o_{e}ntry$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *audit-data-to-entry-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*audit-data-to-entry s pid*)
and *p6*: $t' =$ *fst*(*audit-data-to-entry t pid*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 audit-data-to-entry-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 audit-data-to-entry-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *audit-data-to-entry-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e =$ *AuditEvt*(*Event-audit-data-to-entry pid*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*

and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p2 domain-of-event-def getpid-from-aduit-evevt-def*
by *force*
have *a1*: $s' = fst(audit\text{-}data\text{-}to\text{-}entry\ s\ pid\ )$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(audit\text{-}data\text{-}to\text{-}entry\ t\ pid\ )$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: $pid\ @\ s d$
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 audit-data-to-entry-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *audit-data-to-entry-dwsc-e*: *dynamic-weakly-step-consistent-e* (*AuditEvt*(*Event-audit-data-to-entry pid*))
using *dynamic-weakly-step-consistent-e-def audit-data-to-entry-wsc-e*
by *blast*

29.39.2 proving "audit$_{d}upe_{l}sm_{f}ield$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *audit-dupe-lsm-field-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: $pid\ @\ s\ d$
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(audit\text{-}dupe\text{-}lsm\text{-}field\ s\ pid\ df\ sf)$
and *p6*: $t' = fst(audit\text{-}dupe\text{-}lsm\text{-}field\ t\ pid\ df\ sf)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 audit-dupe-lsm-field-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 audit-dupe-lsm-field-def*

by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *audit-dupe-lsm-field-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: e = *AuditEvt*(*Event-audit-dupe-lsm-field pid df sf*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-aduit-evevt-def*
by *force*
have *a1*: s' = *fst*(*audit-dupe-lsm-field s pid df sf*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*audit-dupe-lsm-field t pid df sf*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 audit-dupe-lsm-field-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *audit-dupe-lsm-field-dwsc-e*: *dynamic-weakly-step-consistent-e* (*AuditEvt*(
Event-audit-dupe-lsm-field pid df sf))
using *dynamic-weakly-step-consistent-e-def audit-dupe-lsm-field-wsc-e*
by *blast*

29.39.3 proving "update$_lsm_r$ule" satisfyingthe" weaklystepconsistent" property

lemma *update-lsm-rule-wsc*:
assumes *p0*: *reachable0 s*

and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(update\text{-}lsm\text{-}rule\ s\ pid\ krule)$
and *p6*: $t' = fst(update\text{-}lsm\text{-}rule\ t\ pid\ krule)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 update-lsm-rule-def*
by *simp*
have *a1* : $t = t'$
using *p6 update-lsm-rule-def*
by *auto*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *update-lsm-rule-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = AuditEvt(\ Event\text{-}audit\text{-}rule\text{-}known\ pid\ krule)$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-aduit-evevt-def*
by *force*
have *a1*: $s' = fst(update\text{-}lsm\text{-}rule\ s\ pid\ krule)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(update\text{-}lsm\text{-}rule\ t\ pid\ krule)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 update-lsm-rule-wsc*

by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *update-lsm-rule-dwsc-e*: *dynamic-weakly-step-consistent-e* (*AuditEvt*(*Event-audit-rule-known pid krule*))
using *dynamic-weakly-step-consistent-e-def update-lsm-rule-wsc-e*
by *blast*

29.39.4 proving "audit$_rule_match$" satisfyingthe" weaklystepconsistent" property

lemma *audit-rule-match-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(audit\text{-}rule\text{-}match\ s\ pid\ sid)$
and *p6*: $t' = fst(audit\text{-}rule\text{-}match\ t\ pid\ sid)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 audit-rule-match-def*
by (*metis fstI*)
have *a1* : $t = t'$
using *p6 audit-rule-match-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *audit-rule-match-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *AuditEvt*(*Event-audit-rule-match pid sid*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*

using *p2 domain-of-event-def getpid-from-aduit-evevt-def*
by *force*
have *a1*: $s' = fst(audit\text{-}rule\text{-}match\ s\ pid\ sid)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(audit\text{-}rule\text{-}match\ t\ pid\ sid)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 audit-rule-match-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *audit-rule-match-dwsc-e*: *dynamic-weakly-step-consistent-e* (*AuditEvt*(*Event-audit-rule-match pid sid*))
using *dynamic-weakly-step-consistent-e-def audit-rule-match-wsc-e*
by *blast*

29.39.5 proving "audit$_{free_lsm_field}$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *audit-free-lsm-field-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(audit\text{-}free\text{-}lsm\text{-}field\ s\ pid\ f)$
and *p6*: $t' = fst(audit\text{-}free\text{-}lsm\text{-}field\ t\ pid\ f)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 audit-free-lsm-field-def*
by *simp*
have *a1* : $t = t'$
using *p6 audit-free-lsm-field-def*
by *auto*
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *audit-free-lsm-field-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *AuditEvt*(*Event-audit-rule-free pid f*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s'* = *exec-event s e*
and *p7*: *t'* = *exec-event t e*
shows *s'* ∼ *d* ∼ *t'*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-aduit-evevt-def*
by *force*
have *a1*: *s'* = *fst*(*audit-free-lsm-field s pid f*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t'* = *fst*(*audit-free-lsm-field t pid f*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s'* ∼ *d* ∼ *t'*
using *a1 a2 a3 a4 p0 p1 p3 p5 p4 audit-free-lsm-field-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *audit-free-lsm-field-dwsc-e*: *dynamic-weakly-step-consistent-e* (*AuditEvt*(*Event-audit-rule-free pid f*))
using *dynamic-weakly-step-consistent-e-def audit-free-lsm-field-wsc-e*
by *blast*

29.40 smack sock hooks weakly step consistent

29.40.1 proving "unix$_s$tream$_c$onnect"satisfyingthe"weaklystepconsistent"property

lemma *unix-stream-connect-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s'* = *fst*(*unix-stream-connect s pid sock uaddr addr-len flags'*)
and *p6*: *t'* = *fst*(*unix-stream-connect t pid sock uaddr addr-len flags'*)

shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 unix-stream-connect-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 unix-stream-connect-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *unix-stream-connect-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = $ *SocketEvt*(*Event-unix-stream-connect pid sock uaddr addr-len flags'*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' = $ *exec-event s e*
and *p7*: $t' = $ *exec-event t e*
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-socket-evevt-def*
by *force*
have *a1*: $s' = $ *fst*(*unix-stream-connect s pid sock uaddr addr-len flags'*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = $ *fst*(*unix-stream-connect t pid sock uaddr addr-len flags'*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 unix-stream-connect-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *unix-stream-connect-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(*Event-unix-stream-connect pid sock uaddr addr-len flags'*))
  using *dynamic-weakly-step-consistent-e-def unix-stream-connect-wsc-e*
  by *blast*

29.40.2 proving "unix$_dgram_connect$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *unix-dgram-connect-wsc*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $s \sim d \sim t$
  and *p3*: *pid* @ *s d*
  and *p4*: $s \sim pid \sim t$
  and *p5*: $s' = fst(unix\text{-}dgram\text{-}connect\ s\ pid\ sock\ uaddr\ alen\ flags')$
  and *p6*: $t' = fst(unix\text{-}dgram\text{-}connect\ t\ pid\ sock\ uaddr\ alen\ flags')$
 shows $s' \sim d \sim t'$
  proof −
 {
  have *a0* : $s = s'$
   using *p5 unix-dgram-connect-def*
   by (*smt fstI*)
  have *a1* : $t = t'$
   using *p6 unix-dgram-connect-def*
   by (*smt fst-conv*)
  have *a2*: $s' \sim d \sim t'$
   using *a0 a1 p2*
   by *blast*
 }
 then show *?thesis* **by** *auto*
qed

lemma *unix-dgram-connect-wsc-e*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: *e* = *SocketEvt*(*Event-unix-dgram-connect pid sock uaddr alen flags'*
)
  and *p3*: $s \sim d \sim t$
  and *p4*: (*the* (*domain-of-event e*)) @ *s d*
  and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
  and *p6*: s' = *exec-event s e*
  and *p7*: t' = *exec-event t e*
 shows $s' \sim d \sim t'$
 proof −
  {
  have *a0* : (*the* (*domain-of-event e*)) = *pid*
   using *p2 domain-of-event-def getpid-from-socket-evevt-def*
   by *force*
  have *a1*: $s' = fst(unix\text{-}dgram\text{-}connect\ s\ pid\ sock\ uaddr\ alen\ flags')$
   using *p2 p6 exec-event-def* **by** *auto*

have *a2*: $t' = fst(unix\text{-}dgram\text{-}connect\ t\ pid\ sock\ uaddr\ alen\ flags')$
  using *p2 p7 exec-event-def*
  by *auto*
have *a3*: *pid @ sd*
  using *p4 a0*
  by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
  by *blast*
have *a5*: $s' \sim d \sim t'$
  using *a1 a2 a3 a4 p0 p1 p3 unix-dgram-connect-wsc*
  by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *unix-dgram-connect-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(*Event-unix-dgram-connect pid sock uaddr alen flags'*))
  using *dynamic-weakly-step-consistent-e-def unix-dgram-connect-wsc-e*
  by *blast*

29.40.3 proving "unix$_{d}gram_{s}endmsg$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *unix-dgram-sendmsg-wsc*:
assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $s \sim d \sim t$
  and *p3*: *pid @ s d*
  and *p4*: $s \sim pid \sim t$
  and *p5*: $s' = fst(unix\text{-}dgram\text{-}sendmsg\ s\ pid\ sock\ uaddr\ alen)$
  and *p6*: $t' = fst(unix\text{-}dgram\text{-}sendmsg\ t\ pid\ sock\ uaddr\ alen)$
shows $s' \sim d \sim t'$
proof −
{
  have *a0* : $s = s'$
    using *p5 unix-dgram-sendmsg-def*
    by (*smt fstI*)
  have *a1* : $t = t'$
    using *p6 unix-dgram-sendmsg-def*
    by (*smt fst-conv*)
  have *a2*: $s' \sim d \sim t'$
    using *a0 a1 p2*
    by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *unix-dgram-sendmsg-wsc-e*:
assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*

and *p2*: *e* = *SocketEvt*(*Event-unix-dgram-sendmsg pid sock uaddr alen*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s′* = *exec-event s e*
and *p7*: *t′* = *exec-event t e*
shows *s′* ∼ *d* ∼ *t′*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-socket-evevt-def*
by *force*
have *a1*: *s′* = *fst*(*unix-dgram-sendmsg s pid sock uaddr alen*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t′* = *fst*(*unix-dgram-sendmsg t pid sock uaddr alen*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s′* ∼ *d* ∼ *t′*
using *a1 a2 a3 a4 p0 p1 p3 unix-dgram-sendmsg-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *unix-dgram-sendmsg-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(
Event-unix-dgram-sendmsg pid sock uaddr alen))
using *dynamic-weakly-step-consistent-e-def unix-dgram-sendmsg-wsc-e*
by *blast*

29.40.4 proving "sys$_b ind'$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *sys-bind′-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s′* = *fst*(*sys-bind′ s pid fd umyaddr addrlen*)
and *p6*: *t′* = *fst*(*sys-bind′ t pid fd umyaddr addrlen*)
shows *s′* ∼ *d* ∼ *t′*
proof −
{
have *a0* : *s* = *s′*
using *p5 sys-bind′-def*

by (*smt fstI*)
have *a1* : $t = t'$
using *p6 sys-bind'-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* by *auto*
qed

lemma *sys-bind'-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: e = *SocketEvt*(*Event-sys-bind' pid fd umyaddr addrlen*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-socket-evevt-def*
by *force*
have *a1*: s' = *fst*(*sys-bind' s pid fd umyaddr addrlen*)
using *p2 p6 exec-event-def* by *auto*
have *a2*: t' = *fst*(*sys-bind' t pid fd umyaddr addrlen*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ using *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 sys-bind'-wsc*
by *blast*
}
then show *?thesis* by *auto*
qed

lemma *sys-bind'-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(*Event-sys-bind' pid fd umyaddr addrlen*))
using *dynamic-weakly-step-consistent-e-def sys-bind'-wsc-e*
by *blast*

29.40.5 proving "sys$_c$*onnect′" satisfyingthe" weaklystepconsistent" property*

lemma *sys-connect′-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*sys-connect′ s pid fd uservaddr addrlen*)
and *p6*: $t' =$ *fst*(*sys-connect′ t pid fd uservaddr addrlen*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 sys-connect′-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 sys-connect′-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sys-connect′-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *SocketEvt*(*Event-sys-connect′ pid fd uservaddr addrlen*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-socket-evevt-def*
by *force*
have *a1*: $s' =$ *fst*(*sys-connect′ s pid fd uservaddr addrlen*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' =$ *fst*(*sys-connect′ t pid fd uservaddr addrlen*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*

by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 sys-connect'-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sys-connect'-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(*Event-sys-connect' pid fd uservaddr addrlen*))
using *dynamic-weakly-step-consistent-e-def sys-connect'-wsc-e*
by *blast*

29.40.6 proving "sock$_s$*endmsg*" *satisfyingthe*" *weaklystepconsistent*" *property*

lemma *sock-sendmsg-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' =$ *fst*(*sock-sendmsg s pid sock msg*)
and *p6*: $t' =$ *fst*(*sock-sendmsg t pid sock msg*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 sock-sendmsg-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 sock-sendmsg-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sock-sendmsg-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e =$ *SocketEvt*(*Event-sock-sendmsg pid sock msg*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: $s' =$ *exec-event s e*
and *p7*: $t' =$ *exec-event t e*
shows $s' \sim d \sim t'$

proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-socket-evevt-def*
by *force*
have *a1*: *s'* = *fst*(*sock-sendmsg s pid sock msg*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t'* = *fst*(*sock-sendmsg t pid sock msg*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s'* ∼ *d* ∼ *t'*
using *a1 a2 a3 a4 p0 p1 p3 sock-sendmsg-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sock-sendmsg-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(*Event-sock-sendmsg pid sock msg*))
using *dynamic-weakly-step-consistent-e-def sock-sendmsg-wsc-e*
by *blast*

29.40.7 proving "sock$_{r}ecvmsg$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *sock-recvmsg-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s'* = *fst*(*sock-recvmsg s pid sock msg flags'*)
and *p6*: *t'* = *fst*(*sock-recvmsg t pid sock msg flags'*)
shows *s'* ∼ *d* ∼ *t'*
proof –
{
have *a0* : *s* = *s'*
using *p5 sock-recvmsg-def*
by *simp*
have *a1* : *t* = *t'*
using *p6 sock-recvmsg-def*
by *auto*
have *a2*: *s'* ∼ *d* ∼ *t'*
using *a0 a1 p2*
by *blast*

}
then show *?thesis* **by** *auto*
qed

lemma *sock-recvmsg-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = SocketEvt($ *Event-sock-recvmsg pid sock msg flags'* $)$
and *p3*: $s \sim d \sim t$
and *p4*: $(the\ (domain\text{-}of\text{-}event\ e))$ @ $s\ d$
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p2 domain-of-event-def getpid-from-socket-evevt-def*
by *force*
have *a1*: $s' = fst(sock\text{-}recvmsg\ s\ pid\ sock\ msg\ flags')$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(sock\text{-}recvmsg\ t\ pid\ sock\ msg\ flags')$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 sock-recvmsg-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sock-recvmsg-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(*Event-sock-recvmsg pid sock msg flags'*))
using *dynamic-weakly-step-consistent-e-def sock-recvmsg-wsc-e*
by *blast*

29.40.8 proving "sk$_{f}ilter_{t}rim_{c}ap$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *sk-filter-trim-cap-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$

and *p5*: $s' = fst(sk\text{-}filter\text{-}trim\text{-}cap\ s\ pid\ sk'\ skb\ cap)$
and *p6*: $t' = fst(sk\text{-}filter\text{-}trim\text{-}cap\ t\ pid\ sk'\ skb\ cap)$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 sk-filter-trim-cap-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 sk-filter-trim-cap-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sk-filter-trim-cap-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = SocketEvt(\ Event\text{-}sk\text{-}filter\text{-}trim\text{-}cap\ pid\ sk'\ skb\ cap\ )$
and *p3*: $s \sim d \sim t$
and *p4*: $(the\ (domain\text{-}of\text{-}event\ e))\ @\ s\ d$
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p2 domain-of-event-def getpid-from-socket-evevt-def*
by *force*
have *a1*: $s' = fst(sk\text{-}filter\text{-}trim\text{-}cap\ s\ pid\ sk'\ skb\ cap)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(sk\text{-}filter\text{-}trim\text{-}cap\ t\ pid\ sk'\ skb\ cap)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: $pid\ @\ s d$
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 sk-filter-trim-cap-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sk-filter-trim-cap-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(*Event-sk-filter-trim-cap pid sk′ skb cap*))
  using *dynamic-weakly-step-consistent-e-def sk-filter-trim-cap-wsc-e*
  by *blast*

29.40.9 proving "sock$_getsockopt"satisfyingthe"weaklystepconsistent"property$

lemma *sock-getsockopt-cap-wsc*:
 assumes *p0*: *reachable0 s*
   and *p1*: *reachable0 t*
   and *p2*: $s \sim d \sim t$
   and *p3*: *pid* @ *s d*
   and *p4*: $s \sim pid \sim t$
   and *p5*: $s' =$ *fst*(*sock-getsockopt s pid sock level′ optname optval optlen*)
   and *p6*: $t' =$ *fst*(*sock-getsockopt t pid sock level′ optname optval optlen*)
  shows $s' \sim d \sim t'$
   proof −
  {
   have *a0* : $s = s'$
     using *p5 sock-getsockopt-def*
     by (*smt fstI*)
   have *a1* : $t = t'$
     using *p6 sock-getsockopt-def*
     by (*smt fst-conv*)
   have *a2*: $s' \sim d \sim t'$
     using *a0 a1 p2*
     by *blast*
  }
  then show *?thesis* **by** *auto*
qed

lemma *sock-getsockopt-wsc-e*:
  assumes *p0*: *reachable0 s*
   and *p1*: *reachable0 t*
   and *p2*: *e* = *SocketEvt*(*Event-sock-getsockopt pid sock level′ optname optval optlen*)
   and *p3*: $s \sim d \sim t$
   and *p4*: (*the* (*domain-of-event e*)) @ *s d*
   and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
   and *p6*: $s' =$ *exec-event s e*
   and *p7*: $t' =$ *exec-event t e*
  shows $s' \sim d \sim t'$
  proof −
   {
   have *a0* : (*the* (*domain-of-event e*)) = *pid*
     using *p2 domain-of-event-def getpid-from-socket-evevt-def*
     by *force*
   have *a1*: $s' =$ *fst*(*sock-getsockopt s pid sock level′ optname optval optlen*)

using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(sock\text{-}getsockopt\ t\ pid\ sock\ level'\ optname\ optval\ optlen)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *s d*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 sock-getsockopt-cap-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sock-getsockopt-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(*Event-sock-getsockopt pid sock level′ optname optval optlen*))
using *dynamic-weakly-step-consistent-e-def sock-getsockopt-wsc-e*
by *blast*

29.40.10 proving "unix$_{g}et_{p}eersec_{d}gram$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *unix-get-peersec-dgram-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(unix\text{-}get\text{-}peersec\text{-}dgram\ s\ pid\ sock\ scm\ )$
and *p6*: $t' = fst(unix\text{-}get\text{-}peersec\text{-}dgram\ t\ pid\ sock\ scm\ )$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 unix-get-peersec-dgram-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 unix-get-peersec-dgram-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *unix-get-peersec-dgram-wsc-e*:
assumes *p0*: *reachable0 s*

and *p1*: *reachable0 t*
and *p2*: *e* = *SocketEvt*(*Event-unix-get-peersec-dgram pid sock scm*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s'* = *exec-event s e*
and *p7*: *t'* = *exec-event t e*
shows *s'* ∼ *d* ∼ *t'*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-socket-evevt-def*
by *force*
have *a1*: *s'* = *fst*(*unix-get-peersec-dgram s pid sock scm*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t'* = *fst*(*unix-get-peersec-dgram t pid sock scm*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s'* ∼ *d* ∼ *t'*
using *a1 a2 a3 a4 p0 p1 p3 unix-get-peersec-dgram-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *unix-get-peersec-dgram-dwsc-e*: *dynamic-weakly-step-consistent-e* (*SocketEvt*(*Event-unix-get-peersec-dgram pid sock scm*))
using *dynamic-weakly-step-consistent-e-def unix-get-peersec-dgram-wsc-e*
by *blast*

29.41 smack inodes hooks weakly step consistent

29.41.1 proving "vfs$_l$*ink*"*satisfyingthe*"*weaklystepconsistent*"*property*

lemma *vfs-link-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s'* = *fst*(*vfs-link s pid old-dentry dir new-dentry delegated-inode*)
and *p6*: *t'* = *fst*(*vfs-link t pid old-dentry dir new-dentry delegated-inode*)
shows *s'* ∼ *d* ∼ *t'*
proof −
{

have *a0* : $s = s'$
using *p5 vfs-link-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 vfs-link-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-link-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *InodeEvt*(*Event-vfs-link pid old-dentry dir new-dentry delegated-inode*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: s' = *fst*(*vfs-link s pid old-dentry dir new-dentry delegated-inode*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*vfs-link t pid old-dentry dir new-dentry delegated-inode*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim$ *pid* $\sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 vfs-link-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-link-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-vfs-link pid old-dentry dir new-dentry delegated-inode*))
using *dynamic-weakly-step-consistent-e-def vfs-link-wsc-e*

by *blast*

29.41.2 proving "vfs$_unlink$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *vfs-unlink-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(vfs\text{-}unlink\ s\ pid\ \ dir\ dentry\ delegated\text{-}inode)$
and *p6*: $t' = fst(vfs\text{-}unlink\ t\ pid\ \ dir\ dentry\ delegated\text{-}inode)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 vfs-unlink-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 vfs-unlink-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-unlink-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = InodeEvt(\ Event\text{-}vfs\text{-}unlink\ pid\ \ dir\ dentry\ delegated\text{-}inode)$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: $s' = fst(vfs\text{-}unlink\ s\ pid\ \ dir\ dentry\ delegated\text{-}inode)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(vfs\text{-}unlink\ t\ pid\ \ dir\ dentry\ delegated\text{-}inode)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*

using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 vfs-unlink-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-unlink-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-vfs-unlink pid dir dentry delegated-inode*))
using *dynamic-weakly-step-consistent-e-def vfs-unlink-wsc-e*
by *blast*

29.41.3 proving "vfs$_r$*mdir*"*satisfyingthe*"*weaklystepconsistent*"*property*

lemma *vfs-rmdir-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(vfs\text{-}rmdir\ s\ pid\ dir\ dentry)$
and *p6*: $t' = fst(vfs\text{-}rmdir\ t\ pid\ dir\ dentry)$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 vfs-rmdir-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 vfs-rmdir-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-rmdir-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *InodeEvt*(*Event-vfs-rmdir pid dir dentry*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$

and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: s' = *fst*(*vfs-rmdir s pid dir dentry*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*vfs-rmdir t pid dir dentry*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 vfs-rmdir-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-rmdir-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-vfs-rmdir pid dir dentry*))
using *dynamic-weakly-step-consistent-e-def vfs-rmdir-wsc-e*
by *blast*

29.41.4 proving "vfs$_{r}ename$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *vfs-rename-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: s' = *fst*(*vfs-rename s pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*)
and *p6*: t' = *fst*(*vfs-rename t pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 vfs-rename-def*
by (*smt fstI*)
have *a1* : $t = t'$

using *p6 vfs-rename-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-rename-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *InodeEvt*(*Event-vfs-rename pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: s' = *fst*(*vfs-rename s pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*vfs-rename t pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 vfs-rename-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-rename-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-vfs-rename pid old-dir old-dentry new-dir new-dentry delegated-inode flgs*))
using *dynamic-weakly-step-consistent-e-def vfs-rename-wsc-e*
by *blast*

29.41.5 proving "inode$_permission" satisfying the "weakly step consistent" property

lemma *inode-permission-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(inode\text{-}permission\ s\ pid\ inode\ mask')$
and *p6*: $t' = fst(inode\text{-}permission\ t\ pid\ inode\ mask')$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 inode-permission-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 inode-permission-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *inode-permission-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = InodeEvt(\ Event\text{-}inode\text{-}permission\ pid\ inode\ mask'\ )$
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: $s' = fst(inode\text{-}permission\ s\ pid\ inode\ mask')$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(inode\text{-}permission\ t\ pid\ inode\ mask')$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*

by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 inode-permission-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *inode-permission-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-inode-permission pid inode mask'*))
using *dynamic-weakly-step-consistent-e-def inode-permission-wsc-e*
by *blast*

29.41.6 proving "notify$_change$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *notify-change-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: s' = *fst*(*notify-change s pid dentry attr' delegated-inode*)
and *p6*: t' = *fst*(*notify-change t pid dentry attr' delegated-inode*)
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 notify-change-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 notify-change-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *notify-change-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *InodeEvt*(*Event-notify-change pid dentry attr' delegated-inode*
)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*

shows $s' \sim d \sim t'$
proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: s' = *fst*(*notify-change s pid dentry attr' delegated-inode*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: t' = *fst*(*notify-change t pid dentry attr' delegated-inode*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 notify-change-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *notify-change-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-notify-change pid dentry attr' delegated-inode*))
using *dynamic-weakly-step-consistent-e-def notify-change-wsc-e*
by *blast*

29.41.7 proving "fat$_ioctl_set_attributes$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *fat-ioctl-set-attributes-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: s' = *fst*(*fat-ioctl-set-attributes s pid f*)
and *p6*: t' = *fst*(*fat-ioctl-set-attributes t pid f*)
shows $s' \sim d \sim t'$
proof –
{
have *a0* : $s = s'$
using *p5 fat-ioctl-set-attributes-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 fat-ioctl-set-attributes-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*

by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *fat-ioctl-set-attributes-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $e = InodeEvt(\ Event\text{-}fat\text{-}ioctl\text{-}set\text{-}attributes\ pid\ \ f)$
and *p3*: $s \sim d \sim t$
and *p4*: $(the\ (domain\text{-}of\text{-}event\ e))\ @\ s\ \ d$
and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and *p6*: $s' = exec\text{-}event\ s\ e$
and *p7*: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: $s' = fst(fat\text{-}ioctl\text{-}set\text{-}attributes\ s\ pid\ f)$
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(fat\text{-}ioctl\text{-}set\text{-}attributes\ t\ pid\ f)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: $pid\ @\ sd$
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 fat-ioctl-set-attributes-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *fat-ioctl-set-attributes-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-fat-ioctl-set-attributes pid f*))
using *dynamic-weakly-step-consistent-e-def fat-ioctl-set-attributes-wsc-e*
by *blast*

29.41.8 proving "vfs$_{g}etattr$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *vfs-getattr-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: $pid\ @\ s\ \ d$

and $p4$: $s \sim pid \sim t$
and $p5$: $s' = fst(vfs\text{-}getattr\ s\ pid\ path)$
and $p6$: $t' = fst(vfs\text{-}getattr\ t\ pid\ path)$
shows $s' \sim d \sim t'$
proof −
{
have $a0$: $s = s'$
using $p5$ $vfs\text{-}getattr\text{-}def$
by (smt $fstI$)
have $a1$: $t = t'$
using $p6$ $vfs\text{-}getattr\text{-}def$
by (smt $fst\text{-}conv$)
have $a2$: $s' \sim d \sim t'$
using $a0$ $a1$ $p2$
by $blast$
}
then show $?thesis$ **by** $auto$
qed

lemma $vfs\text{-}getattr\text{-}wsc\text{-}e$:
assumes $p0$: $reachable0\ s$
and $p1$: $reachable0\ t$
and $p2$: $e = InodeEvt(\ Event\text{-}vfs\text{-}getattr\ pid\ \ path\ \ )$
and $p3$: $s \sim d \sim t$
and $p4$: $(the\ (domain\text{-}of\text{-}event\ e))$ @ $s\ d$
and $p5$: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
and $p6$: $s' = exec\text{-}event\ s\ e$
and $p7$: $t' = exec\text{-}event\ t\ e$
shows $s' \sim d \sim t'$
proof −
{
have $a0$: $(the\ (domain\text{-}of\text{-}event\ e)) = pid$
using $p2$ $domain\text{-}of\text{-}event\text{-}def$ $getpid\text{-}from\text{-}inode\text{-}evevt\text{-}def$
by $force$
have $a1$: $s' = fst(vfs\text{-}getattr\ s\ pid\ path)$
using $p2$ $p6$ $exec\text{-}event\text{-}def$ **by** $auto$
have $a2$: $t' = fst(vfs\text{-}getattr\ t\ pid\ path)$
using $p2$ $p7$ $exec\text{-}event\text{-}def$
by $auto$
have $a3$: pid @ $s\,d$
using $p4$ $a0$
by $blast$
have $a4$: $s \sim pid \sim t$ **using** $p5$ $a0$
by $blast$
have $a5$: $s' \sim d \sim t'$
using $a1$ $a2$ $a3$ $a4$ $p0$ $p1$ $p3$ $vfs\text{-}getattr\text{-}wsc$
by $blast$
}
then show $?thesis$ **by** $auto$

qed

lemma *vfs-getattr-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-vfs-getattr pid path*))
  using *dynamic-weakly-step-consistent-e-def vfs-getattr-wsc-e*
  by *blast*

29.41.9 proving "vfs$_setxattr" satisfying the "weakly step consistent" property$

lemma *vfs-setxattr-wsc*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $s \sim d \sim t$
  and *p3*: *pid* @ *s d*
  and *p4*: $s \sim pid \sim t$
  and *p5*: $s' =$ *fst*(*vfs-setxattr s pid dentry name value size' flgs*)
  and *p6*: $t' =$ *fst*(*vfs-setxattr t pid dentry name value size' flgs*)
 shows $s' \sim d \sim t'$
 proof −
 {
  have *a0* : $s = s'$
   using *p5 vfs-setxattr-def*
   by (*smt fstI*)
  have *a1* : $t = t'$
   using *p6 vfs-setxattr-def*
   by (*smt fst-conv*)
  have *a2*: $s' \sim d \sim t'$
   using *a0 a1 p2*
   by *blast*
 }
 then show *?thesis* **by** *auto*
qed

lemma *vfs-setxattr-wsc-e*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $e =$ *InodeEvt*(*Event-vfs-setxattr pid dentry name value size' flgs*)
  and *p3*: $s \sim d \sim t$
  and *p4*: (*the* (*domain-of-event e*)) @ *s d*
  and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
  and *p6*: $s' =$ *exec-event s e*
  and *p7*: $t' =$ *exec-event t e*
 shows $s' \sim d \sim t'$
 proof −
  {
  have *a0* : (*the* (*domain-of-event e*)) = *pid*
   using *p2 domain-of-event-def getpid-from-inode-evevt-def*
   by *force*
  have *a1*: $s' =$ *fst*(*vfs-setxattr s pid dentry name value size' flgs*)

using *p2 p6 exec-event-def* **by** *auto*
have *a2*: $t' = fst(vfs\text{-}setxattr\ t\ pid\ dentry\ name\ value\ size'\ flgs)$
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid @ sd*
using *p4 a0*
by *blast*
have *a4* : $s \sim pid \sim t$ **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 vfs-setxattr-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-setxattr-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-vfs-setxattr pid dentry name value size' flgs*))
using *dynamic-weakly-step-consistent-e-def vfs-setxattr-wsc-e*
by *blast*

29.41.10 proving "vfs$_getxattr$"$satisfyingthe$"$weaklystepconsistent$"$property$

lemma *vfs-getxattr-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid @ s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(vfs\text{-}getxattr\ s\ pid\ dentry\ name\ value\ size'\ )$
and *p6*: $t' = fst(vfs\text{-}getxattr\ t\ pid\ dentry\ name\ value\ size'\ )$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 vfs-getxattr-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 vfs-getxattr-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-getxattr-wsc-e*:
assumes *p0*: *reachable0 s*

and *p1*: *reachable0 t*
and *p2*: *e* = *InodeEvt*(*Event-vfs-getxattr pid dentry name value size'*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s'* = *exec-event s e*
and *p7*: *t'* = *exec-event t e*
shows *s'* ∼ *d* ∼ *t'*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: *s'* = *fst*(*vfs-getxattr s pid dentry name value size'*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t'* = *fst*(*vfs-getxattr t pid dentry name value size'*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s'* ∼ *d* ∼ *t'*
using *a1 a2 a3 a4 p0 p1 p3 vfs-getxattr-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-getxattr-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-vfs-getxattr pid dentry name value size'*))
using *dynamic-weakly-step-consistent-e-def vfs-getxattr-wsc-e*
by *blast*

29.41.11 proving "vfs$_r$*emovexattr*"*satisfyingthe*"*weaklystepconsistent*"*property*

lemma *vfs-removexattr-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s'* = *fst*(*vfs-removexattr s pid dentry name*)
and *p6*: *t'* = *fst*(*vfs-removexattr t pid dentry name*)
shows *s'* ∼ *d* ∼ *t'*
proof −
{
have *a0* : *s* = *s'*

using *p5 vfs-removexattr-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 vfs-removexattr-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-removexattr-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *InodeEvt*(*Event-vfs-removexattr pid dentry name*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s′* = *exec-event s e*
and *p7*: *t′* = *exec-event t e*
shows $s' \sim d \sim t'$
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: *s′* = *fst*(*vfs-removexattr s pid dentry name*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t′* = *fst*(*vfs-removexattr t pid dentry name*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 vfs-removexattr-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *vfs-removexattr-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-vfs-removexattr pid dentry name*))
using *dynamic-weakly-step-consistent-e-def vfs-removexattr-wsc-e*
by *blast*

29.41.12 proving "xattr$_{g}$$etsecurity$" $satisfyingthe$" $weaklystepconsistent$" $property$

lemma *xattr-getsecurity-wsc*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $s \sim d \sim t$
  and *p3*: *pid* @ *s d*
  and *p4*: $s \sim pid \sim t$
  and *p5*: $s' = fst(xattr\text{-}getsecurity\ s\ pid\ \ inode\ name\ value\ size')$
  and *p6*: $t' = fst(xattr\text{-}getsecurity\ t\ pid\ \ inode\ name\ value\ size')$
 shows $s' \sim d \sim t'$
 proof −
 {
  have *a0* : $s = s'$
   using *p5 xattr-getsecurity-def*
   by (*smt fstI*)
  have *a1* : $t = t'$
   using *p6 xattr-getsecurity-def*
   by (*smt fst-conv*)
  have *a2*: $s' \sim d \sim t'$
   using *a0 a1 p2*
   by *blast*
 }
 then show *?thesis* **by** *auto*
qed

lemma *xattr-getsecurity-wsc-e*:
 assumes *p0*: *reachable0 s*
  and *p1*: *reachable0 t*
  and *p2*: $e = InodeEvt(\ Event\text{-}xattr\text{-}getsecurity\ pid\ \ inode\ name\ value\ size')$
  and *p3*: $s \sim d \sim t$
  and *p4*: (*the* (*domain-of-event e*)) @ *s d*
  and *p5*: $s \sim (the\ (domain\text{-}of\text{-}event\ e)) \sim t$
  and *p6*: $s' = exec\text{-}event\ s\ e$
  and *p7*: $t' = exec\text{-}event\ t\ e$
 shows $s' \sim d \sim t'$
 proof −
  {
  have *a0* : (*the* (*domain-of-event e*)) = *pid*
   using *p2 domain-of-event-def getpid-from-inode-evevt-def*
   by *force*
  have *a1*: $s' = fst(xattr\text{-}getsecurity\ s\ pid\ \ inode\ name\ value\ size')$
   using *p2 p6 exec-event-def* **by** *auto*
  have *a2*: $t' = fst(xattr\text{-}getsecurity\ t\ pid\ \ inode\ name\ value\ size')$
   using *p2 p7 exec-event-def*
   by *auto*
  have *a3*: *pid* @ *sd*
   using *p4 a0*
   by *blast*
  have *a4* : $s \sim pid \sim t$ **using** *p5 a0*

by *blast*
have *a5*: $s' \sim d \sim t'$
using *a1 a2 a3 a4 p0 p1 p3 xattr-getsecurity-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *xattr-getsecurity-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-xattr-getsecurity pid inode name value size'*))
using *dynamic-weakly-step-consistent-e-def xattr-getsecurity-wsc-e*
by *blast*

29.41.13 proving "nfs4$_l$*istxattr*$_n$*fs*4$_l$*abel*" *satisfyingthe*" *weaklystepconsistent*" *property*

lemma *nfs4-listxattr-nfs4-label-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: $s \sim d \sim t$
and *p3*: *pid* @ *s d*
and *p4*: $s \sim pid \sim t$
and *p5*: $s' = fst(nfs4\text{-}listxattr\text{-}nfs4\text{-}label\ s\ pid\ \ inode\ name\ \ size')$
and *p6*: $t' = fst(nfs4\text{-}listxattr\text{-}nfs4\text{-}label\ t\ pid\ \ inode\ name\ \ size')$
shows $s' \sim d \sim t'$
proof −
{
have *a0* : $s = s'$
using *p5 nfs4-listxattr-nfs4-label-def*
by (*smt fstI*)
have *a1* : $t = t'$
using *p6 nfs4-listxattr-nfs4-label-def*
by (*smt fst-conv*)
have *a2*: $s' \sim d \sim t'$
using *a0 a1 p2*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *nfs4-listxattr-nfs4-label-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *InodeEvt*(*Event-nfs4-listxattr-nfs4-label pid inode name size'*)
and *p3*: $s \sim d \sim t$
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: $s \sim$ (*the* (*domain-of-event e*)) $\sim t$
and *p6*: s' = *exec-event s e*
and *p7*: t' = *exec-event t e*
shows $s' \sim d \sim t'$

proof –
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: *s′* = *fst*(*nfs4-listxattr-nfs4-label s pid inode name size′*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t′* = *fst*(*nfs4-listxattr-nfs4-label t pid inode name size′*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s′* ∼ *d* ∼ *t′*
using *a1 a2 a3 a4 p0 p1 p3 nfs4-listxattr-nfs4-label-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *nfs4-listxattr-nfs4-label-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(
Event-nfs4-listxattr-nfs4-label pid inode name size′))
using *dynamic-weakly-step-consistent-e-def nfs4-listxattr-nfs4-label-wsc-e*
by *blast*

29.41.14 proving "sockfs$_listxattr" satisfying the "weakly step consistent" property$

lemma *sockfs-listxattr-wsc*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *s* ∼ *d* ∼ *t*
and *p3*: *pid* @ *s d*
and *p4*: *s* ∼ *pid* ∼ *t*
and *p5*: *s′* = *fst*(*sockfs-listxattr s pid dentry buffer size′*)
and *p6*: *t′* = *fst*(*sockfs-listxattr t pid dentry buffer size′*)
shows *s′* ∼ *d* ∼ *t′*
proof –
{
have *a0* : *s* = *s′*
using *p5 sockfs-listxattr-def*
by (*smt fstI*)
have *a1* : *t* = *t′*
using *p6 sockfs-listxattr-def*
by (*smt fst-conv*)
have *a2*: *s′* ∼ *d* ∼ *t′*
using *a0 a1 p2*
by *blast*

}
then show *?thesis* **by** *auto*
qed

lemma *sockfs-listxattr-wsc-e*:
assumes *p0*: *reachable0 s*
and *p1*: *reachable0 t*
and *p2*: *e* = *InodeEvt*(*Event-sockfs-listxattr pid dentry buffer size'*)
and *p3*: *s* ∼ *d* ∼ *t*
and *p4*: (*the* (*domain-of-event e*)) @ *s d*
and *p5*: *s* ∼ (*the* (*domain-of-event e*)) ∼ *t*
and *p6*: *s'* = *exec-event s e*
and *p7*: *t'* = *exec-event t e*
shows *s'* ∼ *d* ∼ *t'*
proof −
{
have *a0* : (*the* (*domain-of-event e*)) = *pid*
using *p2 domain-of-event-def getpid-from-inode-evevt-def*
by *force*
have *a1*: *s'* = *fst*(*sockfs-listxattr s pid dentry buffer size'*)
using *p2 p6 exec-event-def* **by** *auto*
have *a2*: *t'* = *fst*(*sockfs-listxattr t pid dentry buffer size'*)
using *p2 p7 exec-event-def*
by *auto*
have *a3*: *pid* @ *sd*
using *p4 a0*
by *blast*
have *a4* : *s* ∼ *pid* ∼ *t* **using** *p5 a0*
by *blast*
have *a5*: *s'* ∼ *d* ∼ *t'*
using *a1 a2 a3 a4 p0 p1 p3 sockfs-listxattr-wsc*
by *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *sockfs-listxattr-dwsc-e*: *dynamic-weakly-step-consistent-e* (*InodeEvt*(*Event-sockfs-listxattr pid dentry buffer size'*))
using *dynamic-weakly-step-consistent-e-def sockfs-listxattr-wsc-e*
by *blast*

29.41.15 proving the "dynamic step consistent" property

theorem *dynamic-weakly-step-consistent*:*dynamic-weakly-step-consistent*
proof −
{
fix *e*
have *dynamic-weakly-step-consistent-e e*
apply(*induct e*)

using *sb-copy-data-dwsc-e do-remount-dwsc-e show-sb-opts-dwsc-e statfs-by-dentry-dwsc-e do-mount-dwsc-e do-umount-dwsc-e pivot-root-dwsc-e setup-security-options-dwsc-e*
set-sb-security-dwsc-e nfs-clone-sb-security-dwsc-e parse-security-options-dwsc-e mount-fs-dwsc-e
apply (*metis Event-sb.exhaust*)
using *prepare-creds-dwsc-e*
sys-setreuid-dwsc-e setpgid-dwsc-e do-getpgid-dwsc-e
getsid-dwsc-e getsecid-dwsc-e task-setnice-dwsc-e
set-task-ioprio-dwsc-e
get-task-ioprio-dwsc-e
check-prlimit-permission-dwsc-e
do-prlimit-dwsc-e
task-setscheduler-dwsc-e
task-getscheduler-dwsc-e
task-movememory-dwsc-e
task-kill-dwsc-e
task-prctl-dwsc-e
apply (*metis Event-tsk.exhaust*)
using *ptrace-may-access-dwsc-e ptrace-traceme-dwsc-e Event-Ptrace.exhaust*
apply *metis*
using *check-syslog-permissions-dwsc-e prepare-binprm-dwsc-e*
apply (*metis Event-sys.exhaust*)
using *do-ioctl-dwsc-e syscall-ioctl-dwsc-e*
ksys-ioctl-dwsc-e vm-mmap-pgoff-dwsc-e do-sys-vm86-dwsc-e get-unmapped-area-dwsc-e
validate-mmap-request-dwsc-e generic-setlease-dwsc-e syscall-lock-dwsc-e
do-lock-file-wait-dwsc-e file-fcntl-dwsc-e
file-send-sigiotask-dwsc-e file-receive-dwsc-e do-dentry-open-dwsc-e
apply (*metis Event-file.exhaust*)
using *ipcperms-dwsc-e audit-ipc-obj-dwsc-e*
msg-queue-msgctl-dwsc-e do-msgsnd-dwsc-e msg-queue-msgrcv-dwsc-e
ksys-shmget-dwsc-e sem-msgctl-dwsc-e do-semtimedop-dwsc-e do-shmat-dwsc-e

ksys-semget-dwsc-e shm-msgctl-dwsc-e ksys-msgget-dwsc-e
apply (*metis Event-ipc.exhaust*)
using *key-task-permission-dwsc-e*
keyctl-get-security-dwsc-e **apply** (*metis Event-Key.exhaust*)
using *audit-data-to-entry-dwsc-e audit-dupe-lsm-field-dwsc-e update-lsm-rule-dwsc-e*
audit-rule-match-dwsc-e audit-free-lsm-field-dwsc-e
using *Event-audit.exhaust* **apply** *metis*

using *unix-stream-connect-dwsc-e unix-dgram-connect-dwsc-e unix-dgram-sendmsg-dwsc-e*

sys-bind'-dwsc-e
sys-connect'-dwsc-e sock-sendmsg-dwsc-e sock-recvmsg-dwsc-e sk-filter-trim-cap-dwsc-e

sock-getsockopt-dwsc-e unix-get-peersec-dgram-dwsc-e
apply (*metis Event-network-sock.exhaust*)
using *vfs-link-dwsc-e*

vfs-unlink-dwsc-e
vfs-rmdir-dwsc-e
vfs-rename-dwsc-e
inode-permission-dwsc-e
notify-change-dwsc-e
fat-ioctl-set-attributes-dwsc-e
vfs-getattr-dwsc-e
vfs-setxattr-dwsc-e
vfs-getxattr-dwsc-e
vfs-removexattr-dwsc-e
xattr-getsecurity-dwsc-e
nfs4-listxattr-nfs4-label-dwsc-e
sockfs-listxattr-dwsc-e
by(*metis Event-inode.exhaust*)
}
then show *?thesis*
using *dynamic-weakly-step-consistent-all-evt* **by** *blast*
qed

29.42 Information flow security of linux LSM specification

theorem *noninfluence-sat*: *noninfluence*
using *dynamic-local-respect uc-eq-noninf dynamic-weakly-step-consistent weak-with-step-cons*
by *blast*

theorem *weak-noninfluence-sat*: *weak-noninfluence* **using** *noninf-impl-weak noninfluence-sat*
by *blast*

theorem *nonleakage-sat*: *nonleakage*
using *noninf-impl-nonlk noninfluence-sat* **by** *blast*

theorem *noninterference-r-sat*: *noninterference-r*
using *noninf-impl-nonintf-r noninfluence-sat* **by** *blast*

theorem *noninterference-sat*: *noninterference*
using *noninterference-r-sat nonintf-r-impl-noninterf* **by** *blast*

theorem *weak-noninterference-r-sat*: *weak-noninterference-r*
using *noninterference-r-sat nonintf-r-impl-wk-nonintf-r* **by** *blast*

theorem *weak-noninterference-sat*: *weak-noninterference*
using *noninterference-sat nonintf-impl-weak* **by** *blast*

end
end

29.43 smack$_h$

theory *smack-h*
imports

Main
HOL.Real
HOL.String
../../LSM/Element
begin
typedecl *mutex*
typedecl *list-head*
typedecl *hlist-node*

* Use IPv6 port labeling if IPv6 is enabled and secmarks * are not being used.

definition *SMACK-IPV6-PORT-LABELING* ≡ *1*

definition *SMACK-IPV6-SECMARK-LABELING* ≡ *1*
definition *SMK-LABELLEN* ≡ *24*
definition *SMK-CIPSOLEN* ≡ *24*
definition *SMK-LONGLABEL* ≡ *256*

record *smack-known* =
smk-known :: *string*
smk-secid :: *nat*
smk-rules :: *list-head*
smk-netlabel :: *netlbl-lsm-secattr*

record *superblock-smack* = *smk-root* :: *smack-known*
smk-floor :: *smack-known*
smk-hat :: *smack-known*
smk-default :: *smack-known*
smk-flags :: *int*

definition *SMK-SB-INITIALIZED* ≡ *0x01*
definition *SMK-SB-UNTRUSTED* ≡ *0x02*

record *socket-smack* = *smk-out* :: *smack-known*
smk-in :: *smack-known*
smk-packet :: *smack-known option*

record *inode-smack* = *smk-inode* :: *smack-known*
smk-itask :: *smack-known option*
smk-mmap :: *smack-known option*
smk-lock :: *mutex*
smk-iflags :: *int*
smk-rcu ::*rcu-head*

record *task-smack* = *smk-task* :: *smack-known*
smk-forked :: *smack-known*
smk-rule :: *list-head*

smk-rules-lock :: *mutex*
smk-relabel :: *smack-known list*

record *smack-rule* =
smk-subject :: *smack-known*
smk-object :: *smack-known*
smk-access :: *int*

record *smk-net4addr* = *net4-list* :: *list-head*
net4-smk-host :: *in-addr*
net4-smk-mask :: *in-addr*
net4-smk-masks :: *int*
net4-smk-label :: *smack-known*

record *smk-net6addr* = *list* :: *list-head*
smk-host :: *in6-addr*
smk-mask :: *in6-addr*
smk-masks :: *int*
smk-label :: *smack-known*

typedecl *short*
record *smk-port-label* = *list* :: *list-head*
smk-sock :: *sock*
smk-port :: *nat*
lsmk-in :: *smack-known*
l-smk-out :: *smack-known*
smk-sock-type :: *short*
smk-can-reuse :: *short*

record *smack-known-list-elem* = *list* :: *list-head*
smk-label :: *smack-known*

record *Config-SECURITY-SMACK* = *SECURITY-SMACK* :: *bool*
SECURITY-SMACK-BRINGUP :: *bool*
SECURITY-SMACK-NETFILTER :: *bool*
SECURITY-SMACK-APPEND-SIGNALS :: *bool*
SMACK-IPV6-SECMARK-LABELING :: *bool*
SMACK-IPV6-PORT-LABELING :: *bool*
CONFIG-IPV6 :: *bool*
CONFIG-SECURITY-SMACK-NETFILTER :: *bool*
CONFIG-SECURITY-SMACK-APPEND-SIGNALS :: *bool*

consts *conf* ::*Config-SECURITY-SMACK*

definition *FSDEFAULT-MNT* ≡ *0x01*
definition *FSFLOOR-MNT* ≡ *0x02*
definition *FSHAT-MNT* ≡*0x04*
definition *FSROOT-MNT* ≡ *0x08*
definition *FSTRANS-MNT* ≡ *0x10*
definition *NUM-SMK-MNT-OPTS* ≡ *5*

definition *SMK-FSDEFAULT* ≡''*smackfsdef=*''
definition *SMK-FSFLOOR* ≡''*smackfsfloor=*''
definition *SMK-FSHAT* ≡''*smackfshat=*''
definition *SMK-FSROOT* ≡''*smackfsroot=*''
definition *SMK-FSTRANS* ≡''*smackfstransmute=*''

definition *SMACK-DELETE-OPTION* ≡''−*DELETE*''
definition *SMACK-CIPSO-OPTION* ≡''−*CIPSO*''

definition *SMACK-UNLABELED-SOCKET* ≡ *0*
definition *SMACK-CIPSO-SOCKET* ≡ *1*

definition *SMACK-CIPSO-DOI-DEFAULT* ≡ *3*
definition *SMACK-CIPSO-DOI-INVALID* ≡ −*1*
definition *SMACK-CIPSO-DIRECT-DEFAULT* ≡ *250*
definition *SMACK-CIPSO-MAPPED-DEFAULT* ≡ *251*
definition *SMACK-CIPSO-MAXLEVEL* ≡ *255*
definition *SMACK-CIPSO-MAXCATNUM* ≡ *184*

definition *SMACK-PTRACE-DEFAULT* ≡ *0*
definition *SMACK-PTRACE-EXACT* ≡ *1*
definition *SMACK-PTRACE-DRACONIAN* ≡ *2*
definition *SMACK-PTRACE-MAX* ≡*SMACK-PTRACE-DRACONIAN*

definition *MAY-TRANSMUTE* ≡ *0x00001000*
definition *MAY-LOCK* ≡*0x00002000*
definition *MAY-BRINGUP* ≡*0x00004000*
definition *MAY-DELIVER* ≡ *if CONFIG-SECURITY-SMACK-APPEND-SIGNALS conf then MAY-APPEND else MAY-WRITE*

definition *MAY-ANYREAD* ≡ *bitOR MAY-READ MAY-EXEC*
definition *MAY-NOT* ≡ *0*
definition *MAY-READWRITE* ≡ *bitOR MAY-READ MAY-WRITE*

definition *SMACK-BRINGUP-ALLOW* ≡ *1*
definition *SMACK-UNCONFINED-SUBJECT* ≡ *2*

definition *SMACK-UNCONFINED-OBJECT* ≡ *3*

definition *SMK-INODE-INSTANT* ≡ *1*
definition *SMK-INODE-TRANSMUTE* ≡ *2*
definition *SMK-INODE-CHANGED* ≡ *4*
definition *SMK-INODE-IMPURE* ≡ *8*
definition *TRANS-TRUE-SIZE* ≡ *4*
definition *SMK-CONNECTING* ≡ *0*
definition *SMK-RECEIVING* ≡ *1*
definition *SMK-SENDING* ≡ *2*
consts *smack-known-list* :: *smack-known list*

record *smack-audit-data* = *func* :: *string*
  subject :: *string*
  object :: *string*
  request :: *string*
  result :: *int*
typedecl *common-audit-data*
record *smk-audit-info* = *smk-a* :: *common-audit-data*
  sad :: *smack-audit-data*

definition *smk-of-task* :: *task-smack* ⇒ *smack-known*
  where *smk-of-task tsp* = *smk-task tsp*

definition *smk-of-forked* :: *task-smack* ⇒ *smack-known*
  where *smk-of-forked tsp* = *smk-forked tsp*

definition *SMACK-AUDIT-DENIED* ≡ *0x1*
definition *SMACK-AUDIT-ACCEPT* ≡ *0x2*

end

29.44 smack hooks

theory *SmackHooks*
  imports
    ../../LSM/Element
    ../../LSM/Linux-LSM-Model
    ../../LSM/LSM-Cap
    ../FSP/smack-h
    Main
    HOL.Real
    HOL.String
    HOL−Word.Word-Bitwise
    ../../lib/Monad-WP/NonDetMonadVCG
begin

record *smack-parsed-rule* = *smk-subject* :: *smack-known*
smk-object :: *smack-known*
smk-access1 :: *int*
smk-access2 :: *int*

record *netlbl-audit* = *secid* :: *u32*
loginuid :: *kuid*
sessionid :: *nat*

typedecl *smk-audit-info*
consts *rules* :: *list-head*
consts *nlabel* :: *netlbl-lsm-secattr*

consts *smk-net4addr-list* :: *smk-net4addr list*
consts *smk-net6addr-list* :: *smk-net6addr list*

definition *smack-known-floor* ≡ (|*smk-known* = ''-'',
smk-secid = *5*,
smk-rules = *rules*,
smk-netlabel = *nlabel*|)

definition *smack-known-hat* ≡ (|*smk-known* = ''^'',
smk-secid = *3*,
smk-rules = *rules*,
smk-netlabel = *nlabel* |)

definition *smack-known-huh* ≡ (|*smk-known* = ''?'',
smk-secid = *2*,
smk-rules = *rules*,
smk-netlabel = *nlabel* |)

definition *smack-known-star* ≡ (|*smk-known* = ''*'',
smk-secid = *4*,
smk-rules = *rules* ,
smk-netlabel = *nlabel*|)

definition *smack-known-web* ≡ (|*smk-known* = ''@'',
smk-secid = *7* ,
smk-rules = *rules*,
smk-netlabel = *nlabel*|)

axiomatization *smack-unconfined* :: *smack-known*
where *assumes-unconfined* : *smack-unconfined* ≠ *smack-known-floor* ∧
smack-unconfined ≠ *smack-known-hat* ∧
smack-unconfined ≠ *smack-known-huh* ∧
smack-unconfined ≠ *smack-known-star* ∧
smack-unconfined ≠ *smack-known-web*

record *State'* = *current* :: *process-id*
tasks :: *process-id* $\rightharpoonup$ *Task*
k-superblock :: *t-sb* $\rightharpoonup$ *super-block*
inodes :: *inum* $\rightharpoonup$ *inode*
sdentry :: *dname* $\rightharpoonup$ *dentry*
files :: *fname* $\rightharpoonup$ *Files*
msg-msgs ::*msg-mid* $\rightharpoonup$ *msg-msg*
msg-queues :: *msg-qid* $\rightharpoonup$ *kern-ipc-perm*
keys :: *keyid* $\rightharpoonup$ *key*
sockets :: *socketdesp* $\rightharpoonup$ *socket*
opts :: *opts*

t-security :: *Cred* $\Rightarrow$*task-smack option*
sb-security ::*super-block* $\Rightarrow$*superblock-smack option*
msg-security :: *msg-msg* $\Rightarrow$ *smack-known option*
ipc-security :: *kern-ipc-perm* $\Rightarrow$ *smack-known option*
i-security :: *inode* $\Rightarrow$ *inode-smack option*
f-security :: *Files* $\Rightarrow$ *smack-known option*
sk-security :: *sock* $\Rightarrow$ *socket-smack option*
key-security :: *key* $\Rightarrow$ *smack-known option*
subj-l :: *Subj* $\Rightarrow$*Label*
obj-l :: *Obj* $\Rightarrow$ *Label*
Subjs :: *Subj set*
Objs :: *Obj set*
pol-tab :: (*Subj*,*Obj*) *policy-table*

definition *get-current s* $\equiv$ (*current s*)
definition *get-cur-task s* = *the*(*tasks s* (*get-current s*))

definition *current-cred* :: *Task* $\Rightarrow$ *Cred*
where *current-cred task* = *cred task*

definition *current-real-cred* :: *Task* $\Rightarrow$ *Cred*
where *current-real-cred task* = *real-cred task*

definition *task-cred task* $\equiv$ *cred task*

definition *task-real-cred task* $\equiv$ *real-cred task*

record *Shared* = *smack-enabled* :: *int*
smack-cipso-direct :: *int*
smack-cipso-mapped :: *int*

smack-net-ambient :: *smack-known*
smack-syslog-label :: *smack-known*
smack-ptrace-rule :: *int*
smack-known-lock :: *mutex*
smack-onlycap-lock :: *mutex*

consts *shared* :: *Shared*

definition *string-to-label* :: *string* ⇒ *Label*
where *string-to-label str* ≡ *if str* = *''?'' then Huh*
else if str = *''^'' then Hat*
else if str = *''-'' then Floor*
else if str = *''∗'' then Star*
else if str = *''@'' then Web*
else Normal str

definition *smk-of-subjlabel* :: *State'* ⇒ *process-id*⇒ *Label*
where*smk-of-subjlabel s pid* ≡ *let*
subjlabel = (*t-security s*) (*cred*(*the*(*tasks s pid*))) *in*
if subjlabel = *None then UNDEFINED*
else
string-to-label (*smk-known*(*smk-of-task*(*the*(*t-security s* (*task-cred* (*the*((*tasks s*) *pid*)))))))

definition *smk-of-subjlabel-real* :: *State'* ⇒ *process-id*⇒ *Label*
where*smk-of-subjlabel-real s pid* ≡
string-to-label (*smk-known*(*smk-of-task*(*the*(*t-security s* (*task-real-cred* (*the*((*tasks s*) *pid*)))))))

definition *smk-of-filelabel* :: *State'* ⇒ *Files* ⇒ *Label*
where *smk-of-filelabel s file* ≡ *let flabel* = (*f-security s file*) *in*
if flabel = *None then UNDEFINED*
else
string-to-label(*smk-known* (*the flabel*))

definition *smk-of-ipclabel* :: *State'* ⇒ *kern-ipc-perm* ⇒ *Label*
where *smk-of-ipclabel s ipc'* ≡ *let flabel* = (*ipc-security s ipc'*) *in*
if flabel = *None then UNDEFINED*
else
string-to-label(*smk-known* (*the flabel*))

definition *smk-of-msglabel* :: *State'* ⇒ *msg-msg* ⇒ *Label*
where *smk-of-msglabel s msg'* ≡ *let label* = (*msg-security s msg'*) *in*
if label = *None then UNDEFINED*
else
string-to-label(*smk-known* (*the label*))

definition *smk-of-keylabel* :: *State'* ⇒ *key* ⇒ *Label*
where *smk-of-keylabel s k* ≡ *let label* = (*key-security s k*) *in*

if label = None then UNDEFINED
else
string-to-label(smk-known (the label))

definition *smk-of-sklabel* :: *State′ ⇒ sock ⇒ Label*
where *smk-of-sklabel s k ≡ let label = (sk-security s k) in*
if label = None then UNDEFINED
else
string-to-label(smk-known(smk-in (the label)))

definition *smk-of-inodelabel* :: *State′ ⇒ inode ⇒ Label*
where *smk-of-inodelabel s i ≡ let label = (i-security s i) in*
if label = None then UNDEFINED
else
string-to-label(smk-known(smk-inode (the label)))

definition *smk-of-superblocklabel* :: *State′ ⇒ super-block ⇒ Label*
where *smk-of-superblocklabel s t ≡ let sblabel = (sb-security s t)*
in if sblabel = None then UNDEFINED
else
string-to-label(smk-known (smk-default (the sblabel)))

primrec *smk-of-objectlabel* :: *State′ ⇒ Obj ⇒ Label*
where *smk-of-objectlabel s (File obj) = smk-of-filelabel s obj |*
smk-of-objectlabel s (Sb obj) = smk-of-superblocklabel s obj |
smk-of-objectlabel s (Process obj) = smk-of-subjlabel-real s obj |
smk-of-objectlabel s (IPC obj) = smk-of-ipclabel s obj |
smk-of-objectlabel s (Msg obj) = smk-of-msglabel s obj |
smk-of-objectlabel s (ObjInode obj) = smk-of-inodelabel s obj |
smk-of-objectlabel s (ObjSock obj) = smk-of-sklabel s obj |
smk-of-objectlabel s (ObjKey obj) = smk-of-keylabel s obj

definition *objlabelAccess* :: *Label ⇒ access set*
where *objlabelAccess obj ≡ case obj of Floor ⇒ {READ,EXECUTE} |*
Star ⇒ {READ,EXECUTE,WRITE,APPEND,T,
LOCK } |
- ⇒ {}

definition *smk-access-rules′* :: *State′⇒Label ⇒ Label ⇒ access set*
where *smk-access-rules′ s subj obj ≡*
case subj of Star ⇒ {} |
Hat ⇒ {READ,EXECUTE} |
Floor ⇒ objlabelAccess obj |
Huh ⇒ objlabelAccess obj |
Web ⇒ objlabelAccess obj |
Normal x ⇒ if obj = Floor then objlabelAccess obj
else if obj = Star then objlabelAccess obj

*else if obj = Normal x then {READ,EXECUTE,WRITE,APPEND,T,
LOCK }*
else {}

definition *Label-to-string* :: *Label* ⇒ *string*
where *Label-to-string label′* ≡ *SOME x. Normal x = label′*

fun *user-define-rule* :: *string* ⇒ *string* ⇒ *access set*
where *user-define-rule - - = {}*

definition *smk-access-rules* :: *Label* ⇒ *Label* ⇒ *access set*
where *smk-access-rules subj obj* ≡
if obj = UNDEFINED then {}
else
if subj = Star then {}
else
*if obj = Web ∨ subj = Web then {READ,EXECUTE,WRITE,APPEND,T,
LOCK }*
else
if obj = Star then {READ,EXECUTE,WRITE,APPEND,T, LOCK }
else
if subj = obj then {READ,EXECUTE,WRITE,APPEND,T, LOCK }
else
if obj = Floor ∨ subj = Hat then {READ,LOCK,EXECUTE }
else user-define-rule (Label-to-string subj) (Label-to-string obj)

definition *ReferenceMonitor* :: *State′* ⇒ *Subj* ⇒ *Obj* ⇒ *Request* ⇒ *decision*
where *ReferenceMonitor s subj obj r* ≡
*if (access-rl r) ∈ (smk-access-rules) (smk-of-subjlabel s subj) (smk-of-objectlabel
s obj)*
then allow
else deny

definition *task-security s t* ≡ *the (t-security s (cred t))*
definition *task-real-security s t* ≡ *the (t-security s (real-cred t))*

definition *inode-security s inode = the(i-security s inode)*

definition *get-pid s task* ≡ *SOME pid . (tasks s) pid = Some task*

definition *get-inum s inode* ≡ *SOME inum . (inodes s) inum = Some inode*

definition *get-sbnum s sb* ≡ *SOME i . (k-superblock s) i= Some sb*

definition *smk-of-task-struct* :: *State′* ⇒ *Task* ⇒ *smack-known*
where *smk-of-task-struct s t* ≡ *smk-of-task (task-security s t)*

definition *current-task s* = *the*((*tasks s*)(*current s*))

definition *current-security s* = *task-security s* (*current-task s*)

definition *smk-of-current* :: *State'* ⇒ *smack-known*
where*smk-of-current s* ≡ *smk-of-task*(*task-security s* (*current-task s*))

definition *smk-inode-transmutable* :: *State'* ⇒ *inode* ⇒ *int*
where *smk-inode-transmutable s isp* ≡
let sip = (*the*(*i-security s isp*)) *in*
if (*smk-iflags sip AND SMK-INODE-TRANSMUTE*) ≠ *0 then 1*
else 0

definition *smk-of-inode* :: *State'* ⇒ *inode* ⇒ *smack-known*
where *smk-of-inode s inode* ≡ *smk-inode*(*inode-security s inode*)

definition *smk-bu-note* :: *State'* ⇒ *string* ⇒*smack-known* ⇒*smack-known* ⇒ *int* ⇒ *int* ⇒ *int*
where *smk-bu-note s note sskp oskp m rc* ≡
if (*SECURITY-SMACK-BRINGUP conf*) *then 0*
else if rc ≤ *0 then rc else 0*

definition *smk-bu-current* :: *State'* ⇒ *string* ⇒*smack-known* ⇒ *int* ⇒ *int* ⇒ *int*
where *smk-bu-current s note oskp m rc* ≡
if (*SECURITY-SMACK-BRINGUP conf*) *then 0*
else if rc ≤ *0 then rc else 0*

definition *smk-bu-task* :: *State'* ⇒ *Task* ⇒ *int* ⇒ *int* ⇒ *int*
where *smk-bu-task s otp m rc* ≡
if (*SECURITY-SMACK-BRINGUP conf*) *then*
if rc ≤ *0 then rc*
else
if rc >*SMACK-UNCONFINED-OBJECT then 0*
else rc
else rc

definition *smk-bu-inode* :: *State'* ⇒ *inode* ⇒ *int* ⇒ *int* ⇒ *int*
where *smk-bu-inode s inode m rc* ≡ *if* (*SECURITY-SMACK-BRINGUP conf*) *then 0 else rc*

definition *smk-bu-file* :: *State'* ⇒ *Files* ⇒ *int* ⇒ *int* ⇒ *int*
where *smk-bu-file s f m rc* ≡ *if* (*SECURITY-SMACK-BRINGUP conf*) *then 0 else rc*

definition *smk-bu-credfile* :: *State'* ⇒ *Cred* ⇒*Files* ⇒ *int* ⇒ *int* ⇒ *int*
where *smk-bu-credfile s cred' f m rc* ≡ *if* (*SECURITY-SMACK-BRINGUP conf*) *then 0 else rc*

definition *smack-privileged-cred* :: *int* ⇒ *Cred* ⇒ *bool*
  where *smack-privileged-cred cap c* ≡ *False*

term *the*((*tasks s*) (*current s*))
definition *smack-privileged* :: *State'* ⇒*int* ⇒ *bool*
  where *smack-privileged s cap* ≡
      if flags (*the*((*tasks s*) (*current s*))) = *PF-KTHREAD then True*
      else smack-privileged-cred cap (*current-cred* (*the*((*tasks s*) (*current s*))))

definition *d-backing-inode* :: *State'* ⇒ *dentry* ⇒ *inode option*
  where *d-backing-inode s upper* ≡((*inodes s*)(*d-inode upper*))

definition *get-inode s inum* = *inodes s inum*

definition *get-dentry s dname* ≡ *sdentry s dname*

definition *file-inode* :: *Files* ⇒ *inode*
  where *file-inode f* ≡ *f-inode f*

type-synonym *word32* = *32 word*
type-synonym *word8* = *8 word*
type-synonym *byte* = *word8*

lemma (*PTRACE-MODE-READ AND PTRACE-MODE-ATTACH*) = (*0x00* :: *byte*)
  apply(*simp add*:*PTRACE-MODE-READ-def PTRACE-MODE-ATTACH-def*)
  done

term (*PTRACE-MODE-READ AND PTRACE-MODE-ATTACH*)::*'a*::*len word*

term *sint* (*PTRACE-MODE-READ AND PTRACE-MODE-ATTACH*)
lemma *sint* (*PTRACE-MODE-READ AND PTRACE-MODE-ATTACH*) = *0*
  by(*simp add*:*PTRACE-MODE-READ-def PTRACE-MODE-ATTACH-def*)

consts *smack-rules* :: *smack-rule list*

definition *smk-access-entry* :: *State'* ⇒*string* ⇒ *string* ⇒ *list-head* ⇒(*State'*, *int*) *nondet-monad*
  where *smk-access-entry s subj obj r* = *do*
      may ← *return*(−*ENOENT*);
      may ← *return*((*if* ((*may AND MAY-WRITE*) = *MAY-WRITE*) *then* (*may OR MAY-LOCK*)
          else ((*may*))));
      return may

od

definition *smk-access-out-audit* :: *smack-known* ⇒ *smack-known* ⇒ *int* ⇒ *int*
where *smk-access-out-audit subj obj rc* ≡
if (*SECURITY-SMACK-BRINGUP conf*) ∧ *rc* < *0 then*
let rc = *if obj* = *smack-unconfined then SMACK-UNCONFINED-OBJECT else rc*;
rc = *if subj* = *smack-unconfined then SMACK-UNCONFINED-SUBJECT else rc*
in rc
else rc

definition *smk-access* :: *State'* ⇒ *smack-known* ⇒ *smack-known* ⇒ *int* ⇒ *smk-audit-info option* ⇒ (*State'*, *int*) *nondet-monad*
where *smk-access s subj obj requests a* ≡
do

rc←(*if subj* = *smack-known-star then*
let rc = −*EACCES*
in return(*smk-access-out-audit subj obj rc*)
else
if obj =*smack-known-web* ∨ *subj* = *smack-known-web*
then return(*smk-access-out-audit subj obj 0*)
else
if obj = *smack-known-star then*
return(*smk-access-out-audit subj obj 0*)
else
if smk-known subj = *smk-known obj then*
return(*smk-access-out-audit subj obj 0*)
else if (*requests AND MAY-ANYREAD* = *requests*) ∨ (*requests AND MAY-LOCK* = *requests*)
then return(*smk-access-out-audit subj obj 0*)
else do
may← *smk-access-entry s* (*smk-known subj*) (*smk-known obj*) (*smk-rules subj*);
if may ≤ *0* ∨ (*requests AND may*) ≠ *requests then*
return(*smk-access-out-audit subj obj* (−*EACCES*))
else if (*SECURITY-SMACK-BRINGUP conf*) ∧ (*may AND MAY-BRINGUP* ≠ *0*) *then*
return(*smk-access-out-audit subj obj SMACK-BRINGUP-ALLOW*)
else
return(*smk-access-out-audit subj obj 0*)
od);
return rc
od

definition *smk-tskacc* :: *State′* ⇒ *task-smack* ⇒ *smack-known* ⇒ *int*
⇒ *smk-audit-info* ⇒ (*State′*, *int*) *nondet-monad*
where *smk-tskacc s tsp obj m a* ≡
do
sbj-known ← *return* (*smk-of-task tsp*);
ad ← *return* (*Some a*);
rc ← *smk-access s sbj-known obj m ad*;
rc ← (*if rc* ≥ *0 then*
do may ← *smk-access-entry s* (*smk-known sbj-known*) (*smk-known obj*)
(*smk-rule tsp*) ;
rc′← (*if may* < *0* ∨ (*m AND may*) = *m then return rc*
else return(−*EACCES*)
);
return rc′
od
else return rc);
rc ← (*if rc* ≠ *0* ∧ (*smack-privileged s CAP-MAC-OVERRIDE*) *then return*
0
else return rc
);
return rc
od

definition *smk-curacc* :: *State′* ⇒*smack-known* ⇒ *int* ⇒*smk-audit-info*⇒ (*State′*,
int) *nondet-monad*
where *smk-curacc s obj m a* ≡
do
rc ← *smk-tskacc s* (*current-security s*) *obj m a*;
return rc
od

definition *new-task-smack* :: *smack-known* ⇒ *smack-known* ⇒*nat* ⇒ *task-smack*
option
where *new-task-smack task forked gfp′* ≡
(*SOME t*. ∀ *rule m label* .*if t* = *None then t* = *None*
else t = *Some* (⦇*smk-task* = *task*,
smk-forked = *forked*,
smk-rule = *rule*,
smk-rules-lock = *m*,
smk-relabel = *label*⦈))

definition *new-inode-smack* :: *smack-known* ⇒ *inode-smack option*
where *new-inode-smack skp* ≡
(*SOME t*. ∃ *mp lock forked rcu* .*if t* = *None then t* = *None*
else t = *Some* (⦇*smk-inode* = *skp*,

$smk\text{-}itask = forked$,
$smk\text{-}mmap = mp$,
$smk\text{-}lock = lock$,
$smk\text{-}iflags = 0$,
$smk\text{-}rcu = rcu\|$))

definition *smk-copy-rules* :: *State'* ⇒*list-head* ⇒ *list-head* ⇒*nat*⇒ (*State'*, *int*) *nondet-monad*
  where *smk-copy-rules s nhead ohead g* ≡
      do
        rc ← *return*(*0*);
        return rc
      od

definition *smk-copy-relabel* :: *State'* ⇒ *smack-known list* ⇒ *smack-known list* ⇒ *nat* ⇒ (*State'*, *int*) *nondet-monad*
  where *smk-copy-relabel s nhead ohead g* ≡
      do
        rc ← *return*(*0*);
        return rc
      od

definition *smack-from-secid* :: *u32* ⇒ (*State'*,*smack-known option*) *nondet-monad*
  where *smack-from-secid secid'* ≡
  do
    a' ← *return*(*0*);
    (*a'*, *result*) ← *whileLoop*
    (λ(*a'*, *result*) *secid'*. *a'* < *length*(*smack-known-list*))
    (λ(*a'* ,*result*) . ((*if smk-secid* (*smack-known-list* ! *a'*) = *secid'*
                    then return (*a'* + *1*, *Some* ((*smack-known-list* ! *a'*)))
                    else return (*a'* + *1*, *Some smack-known-huh*))))
                        (*a'*, *Some smack-known-huh*);
  return result
  od

consts *smack-known-hash* :: *smack-known list*

definition *smk-find-entry* :: *string* ⇒ (*State'*,*smack-known option*) *nondet-monad*
  where *smk-find-entry str* ≡
  do
    a' ← *return*(*0*);
    (*a'*, *result*) ← *whileLoop*
    (λ(*a'*, *result*) *b'*. *a'* < *length*(*smack-known-list*))
    (λ(*a'* ,*result*) . ((*if smk-known* (*smack-known-hash* ! *a'*) = *str*
                    then return (*a'* + *1*, *Some* ((*smack-known-list* ! *a'*)))
                    else return (*a'* + *1*, *None*))))
                        (*a'*, *None*);

return result
od

definition *SOCKET-I′* :: *inode* ⇒ *socket-alloc*
where *SOCKET-I′ i* ≡ *SOME sk. skvfs-inode sk* = *i*

definition *SOCKET-I* :: *inode* ⇒ *socket*
where *SOCKET-I i* ≡ *socket* (*SOCKET-I′ i*)

definition *smk-ptrace-mode* :: *mode* ⇒ *int*
where *smk-ptrace-mode m* ≡
if (*m AND PTRACE-MODE-ATTACH*) ≠ *0*
then MAY-READWRITE
else
if (*m AND PTRACE-MODE-READ*) ≠ *0*
then MAY-READ
else 0

definition *smk-ptrace-rule-check* :: *State′* ⇒ *Task* ⇒ *smack-known* ⇒ *nat* ⇒ *string* ⇒ (*State′*, *int*) *nondet-monad*
where *smk-ptrace-rule-check s tracer tracee-known m func′* ≡ *do*
tracercred ← *return*(*task-cred tracer*);
tsp ← *return*(*the*(*t-security s tracercred*));
tracer-known ← *return*(*smk-of-task tsp*);
saip ← (*if* (*int m AND PTRACE-MODE-NOAUDIT*) ≠*0 then*
return (*SOME x*::*smk-audit-info option* . *True*)
else
return (*None*)
);
rc ← (*if* (((*int m*) *AND PTRACE-MODE-ATTACH*) ≠ *0*) ∧ (
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-EXACT*) ∨
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-DRACONIAN*))
then
if smk-known tracer-known = *smk-known tracee-known*
then return 0
else
if (*smack-ptrace-rule shared*) = *SMACK-PTRACE-DRACONIAN*

then return (−*EACCES*)
else
if smack-privileged-cred CAP-SYS-PTRACE tracercred
then return 0
else return (−*EACCES*)
else do
rc ← *smk-tskacc s tsp tracee-known* (*smk-ptrace-mode m*)

$(the\ saip)$;
return rc
od
);
return rc
od

definition *smack-ptrace-access-check* :: *State'* ⇒ *Task* ⇒ *nat* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-ptrace-access-check s ctp m* ≡
do
skp ← *return*(*smk-of-task-struct s ctp*);
r ← *smk-ptrace-rule-check s* (*current-task s*) *skp m ''smack-ptrace-access-check''*;
return(*r*)
od

definition *smack-ptrace-traceme* :: *State'* ⇒ *Task* ⇒(*State'*, *int*) *nondet-monad*
where *smack-ptrace-traceme s ptp* ≡
do
rc ← *return*(*SOME x*:: *int* .*True*);
skp ← *return* (*smk-of-current s*);
rc ←*smk-ptrace-rule-check s ptp skp PTRACE-MODE-ATTACH ''smack-ptrace-traceme''*;
return (*rc*)
od

definition *smack-syslog* :: *State'* ⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where*smack-syslog s typefrom*≡
do
skp ←*return*(*smk-of-current s*);
slabel ← *return*(*smack-syslog-label shared*);
rc ← (*if smack-privileged s CAP-MAC-OVERRIDE*
then return 0
else
if slabel ≠ *skp*
then return (*uminus EACCES*)
else return 0
);
return(*rc*)
od

term *pol-tab s*
term (*pol-tab s c*)((*c*,*t*) := *a*)
term *sorted-list-of-set*
term *SOME ta* . ∀ *p obj*. *p* ∈ *taskset* ∧ *tab* = *tab*((*p*,*obj*):= {}) ∧ *ta* =*ta*(*p*:=*tab*)

term *ta*(*p*:= *SOME tab* . ∀ *p obj*. *p* ∈ *taskset* ∧ *tab* = *tab*((*p*,*obj*):= {}))

definition *cursp* :: *State'* ⇒ *process-id list*
  where *cursp s* ≡ *sorted-list-of-set* {*t* .∀ *p* .*tasks s* = (*tasks s*)(*t* := *Some p*) }

definition *createObjChgTab* :: *State'* ⇒ *Subj* ⇒ *Obj* ⇒(*Subj*,*Obj*) *policy-table*
  where *createObjChgTab s subj object'* ≡
    let taskset = {*t* .∀ *sb* .*tasks s* = (*tasks s*)(*t* := *Some sb*) };
      subjlabel = *smk-of-subjlabel s subj*;
      objlabel = *smk-of-objectlabel s object'*;
      right = *smk-access-rules subjlabel objlabel*;
     tab = *SOME tab* . ∀ *p* . *p* ∈ *taskset* ∧ *tab* = *tab*((*p*,*object'*):= *right*)
    in
     SOME ta . ∀ *p*. *p* ∈ *taskset* ∧ *ta* =*ta*(*p*:=*tab*)

definition *update-access-tab* :: *State'* ⇒ *process-id* ⇒ *Obj*⇒ *State'*
  where *update-access-tab s subj obj* ≡
     let tab = (*pol-tab s*);
     subjlabel = (*smk-of-subjlabel s subj*);
     objectlabel = (*smk-of-objectlabel s obj*);
     right = (*smk-access-rules subjlabel objectlabel*);
     access = ((*pol-tab s subj*)((*subj*,*obj*) := *right*))
     in s(|*pol-tab* := (*pol-tab s*)(*subj* :=*access*)|)

definition *update* :: *State'* ⇒ *Obj* ⇒ (*State'*, *nat*) *nondet-monad*
  where *update s obj* ≡
  do
   a' ← *return*(*0*);
   (*a'*, *result*) ← *whileLoop*
   (λ(*a'*, *result*) *s*. *a'* < *length*(*cursp s*))
   (λ(*a'* ,*result*) . (*return* (*a'*+*1*,*update-access-tab s* (*cursp s* ! *a'*) *obj*)))
            (*a'*, *s*);
  return a'
  od

29.45 Superblock Hooks

definition *smack-sb-alloc-security* :: *State'* ⇒ *super-block* ⇒ (*State'*, *int*) *nondet-monad*
  where*smack-sb-alloc-security s sb* ≡
    do
    sbsp ←*return*(*SOME x* :: *superblock-smack option*. *True*);
    rc ← (*if sbsp* = *None*
       then return (*uminus ENOMEM*)
       else do
         sbsp←*return*((|*smk-root* = *smack-known-floor*,
              smk-floor = *smack-known-floor*,

$smk\text{-}hat = smack\text{-}known\text{-}hat,$
$smk\text{-}default = smack\text{-}known\text{-}floor,$
$smk\text{-}flags = 0$ ⦈
);
$modify(\lambda s .s(|sb\text{-}security := (sb\text{-}security\ s)(sb := Some\ sbsp\ )|));$
$return\ 0$
$od);$
$return(rc)$
od

definition $smack\text{-}sb\text{-}free\text{-}security :: State' \Rightarrow super\text{-}block \Rightarrow (State', unit)\ nondet\text{-}monad$
where $smack\text{-}sb\text{-}free\text{-}security\ s\ sb \equiv do$
$modify(\lambda s .s(|sb\text{-}security := (sb\text{-}security\ s)(sb := None\ )|));$
$return()$
od

definition $smack\text{-}sb\text{-}copy\text{-}data :: State' \Rightarrow string \Rightarrow string \Rightarrow (State', int)\ nondet\text{-}monad$
where $smack\text{-}sb\text{-}copy\text{-}data\ s\ orig\ smackopts \equiv do$
$otheropts \leftarrow return(SOME\ x :: string.\ True);$
$r \leftarrow (if\ length(otheropts) = 0$
$then\ return\ (uminus\ ENOMEM)$
$else\ return\ 0);$
$return(r)$
od

definition $smack\text{-}parse\text{-}opts\text{-}str :: State' \Rightarrow string \Rightarrow opts \Rightarrow (State', int)\ nondet\text{-}monad$
where $smack\text{-}parse\text{-}opts\text{-}str\ s\ options\ opt \equiv do$
$r \leftarrow (if\ length(options) = 0\ then\ return\ 0\ else\ return(uminus\ ENOMEM));$
$return(r)$
od

definition $smack\text{-}set\text{-}mnt\text{-}opts :: State' \Rightarrow super\text{-}block \Rightarrow opts \Rightarrow nat \Rightarrow nat \Rightarrow (State', int)\ nondet\text{-}monad$
where $smack\text{-}set\text{-}mnt\text{-}opts\ s\ sb\ opt\ kern\text{-}flags\ set\text{-}kern\text{-}flags \equiv do$

$root \leftarrow return(s\text{-}root\ sb);$
$inode \leftarrow return(d\text{-}backing\text{-}inode\ s\ (the((sdentry\ s)\ root)));$
$sp \leftarrow return(the(sb\text{-}security\ s\ sb));$
$num\text{-}opts \leftarrow return\ (num\text{-}mnt\text{-}opts\ opt);$
$rc \leftarrow (if\ (smk\text{-}flags\ sp\ AND\ SMK\text{-}SB\text{-}INITIALIZED) \neq 0$
$then\ return\ 0$
$else\ if\ \neg(smack\text{-}privileged\ s\ CAP\text{-}MAC\text{-}ADMIN) \wedge num\text{-}opts \neq 0$
$then\ return\ (-EPERM)$
$else\ return\ 0$
);
$return(rc)$
od

definition *get-ret s m= fst(the-run-state m s)*

definition *get-security-mnt-opts* :: *State′* ⇒ *opts*
where *get-security-mnt-opts s* ≡ *opts s*

definition *smack-sb-kern-mount* :: *State′* ⇒ *super-block*⇒ *int* ⇒ *string*⇒ (*State′*, *int*) *nondet-monad*
where *smack-sb-kern-mount s sb f data* ≡ *do*
options ← (*return data*);
rc ←(*if length*(*data*) = *0 then* (*smack-set-mnt-opts s sb* (*get-security-mnt-opts s*) *0 0*)
else do
rc ← *smack-parse-opts-str s options* (*opts s*);
rc← (*if rc* = *0 then return rc*
else (*smack-set-mnt-opts s sb* (*get-security-mnt-opts s*) *0 0*)
);
return rc
od
);
return(*rc*)
od

definition *smack-sb-statfs* :: *State′* ⇒ *dentry*⇒ (*State′*, *int*) *nondet-monad*
where *smack-sb-statfs s d* ≡ *do*
sbp ← *return*(*the* (*sb-security s* (*d-sb d*)));
ad ← *return* (*SOME x* :: *smk-audit-info .True*);
rc ← *smk-curacc s* (*smk-floor sbp*) *MAY-READ ad*;
rc ← *return*(*smk-bu-current s ′′statfs′′* (*smk-floor sbp*) *MAY-READ rc*);
return(*rc*)
od

29.46 BPRM hooks

definition *ptrace-parent* :: *State′* ⇒*Task* ⇒ *Task option*
where *ptrace-parent s tsk′* ≡ *if unlikely* (*ptrace tsk′*) *then Some* (*the*((*tasks s*) (*parent tsk′*))) *else None*

definition *smack-bprm-set-creds* :: *State′* ⇒ *linux-binprm* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-bprm-set-creds s bprm* ≡ *do*
inode ← *return*(*file-inode*(*lfiles bprm*));
bsp ← *return* (*the*((*t-security s*) (*lcred bprm*)));
rc ← (*if called-set-creds bprm* ≠ *0 then return 0 else*
do
isp ← *return* (*the*(*i-security s inode*));
if (*the*(*smk-itask isp*)) = (*smk-task bsp*) *then return 0 else*
do
sbsp ← *return*(*the*((*sb-security s*)(*i-sb inode*)));
if ((*smk-flags sbsp*) *AND SMK-SB-UNTRUSTED*) ≠ *0* ∧

(*the*(*smk-itask isp*) $\neq$ *smk-root sbsp*)
then return 0 else
if (*unsafe bprm AND LSM-UNSAFE-PTRACE*) $\neq$ *0 then*
do
rc $\leftarrow$ *return 0*;
tracer $\leftarrow$ *return*(*ptrace-parent s* (*get-cur-task s*));
rc $\leftarrow$ (*if tracer* $\neq$ *None then*
do
rc $\leftarrow$ *smk-ptrace-rule-check s* (*the tracer*) (*the*(*smk-itask isp*))
PTRACE-MODE-ATTACH ''smack-bprm-set-creds'';
return rc
od
else return rc);
if rc $\neq$ *0 then return rc*
else do
modify(λs .*s*(|*t-security* :=
(*t-security s*)((*lcred bprm*) :=
Some(*bsp*(|*smk-task*:= *the*(*smk-itask isp*)|)))|)
);
return 0
od
od
else if (*unsafe bprm*) $\neq$ *0 then return* ($-$*EPERM*)
else
return 0
od
od
);
return(*rc*)
od

29.47 inode hooks

definition *smack-inode-alloc-security* :: *State'* $\Rightarrow$ *inode* $\Rightarrow$ (*State'*, *int*) *nondet-monad*
where *smack-inode-alloc-security s inode* $\equiv$ *do*
skp $\leftarrow$ (*return* (*smk-of-current s*));
i-s $\leftarrow$ *return*(*new-inode-smack skp*);
modify(λs .*s*(|*i-security* := (*i-security s*)(*inode* := *i-s*)|));
rc $\leftarrow$(*if* (*i-security s inode*)= *None*
then return (*uminus ENOMEM*)
else return 0
);
return(*rc*)
od

definition *smack-inode-free-security* :: *State'* $\Rightarrow$ *inode* $\Rightarrow$ (*State'*, *unit*) *nondet-monad*
where *smack-inode-free-security s inode* $\equiv$ *do*
modify(λs .*s*(|*i-security* := (*i-security s*)(*inode* := *None*)|));

return ()
od

definition *smack-inode-init-security* :: *State'* ⇒ *inode* ⇒ *inode* ⇒ *string* ⇒ *string* ⇒ *string* ⇒*int*⇒ (*State'*, *int*) *nondet-monad*
where *smack-inode-init-security s inode dir qstr name value len'* ≡ *do*
skp ← (*return* (*smk-of-current s*));
issp ← (*return* (*the*(*i-security s inode*)));
isp ← *return*(*smk-of-inode s inode*);
dsp ← *return*(*smk-of-inode s dir*);
rc ←(*if length*(*value*) ≠ *0* ∧ *len'* ≠ *0 then*
do
may ← *smk-access-entry s* (*smk-known skp*) (*smk-known dsp*) (*smk-rules skp*);
rc ← (*if* ((*may* > *0* ∧ (*may AND MAY-TRANSMUTE*) ≠ *0*) ∧ (*smk-inode-transmutable s dir*) ≠ *0*)
then do
f ← *return* (*bitOR* (*smk-iflags issp*) *SMK-INODE-CHANGED*);
modify(λ*s* .*s*⦇*i-security* := (*i-security s*)(*inode* := *Some* (*issp*⦇*smk-iflags* := *f* ⦈))⦈);
value ← *return*(*smk-known dsp*);
if length(*value*) = *0*
then *return* (*uminus ENOMEM*)
else return 0
od
else
if length(*smk-known isp*) = *0*
then *return* (*uminus ENOMEM*)
else return 0
);
return (*rc*)
od
else return 0);
return(*rc*)
od

definition *smack-inode-link* :: *State'* ⇒ *dentry* ⇒ *inode* ⇒ *dentry* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-inode-link s old dir new* ≡ *do*
isp ← (*return* (*smk-of-inode s* (*the*(*d-backing-inode s old*))));
ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);
rc ← *return* (*smk-bu-inode s* (*the*(*d-backing-inode s old*)) *MAY-WRITE* ((*get-ret s* (*smk-curacc s isp MAY-WRITE ad*))));
rc ← (*if rc* = *0* ∧ *d-is-positive new*
then return (*smk-bu-inode s* (*the*(*d-backing-inode s new*)) *MAY-WRITE*

((*get-ret s* (*smk-curacc s isp MAY-WRITE ad*))))
else return rc);
return(*rc*)
od

definition *smack-inode-unlink* :: *State'* ⇒ *inode* ⇒*dentry*⇒ (*State'*, *int*) *nondet-monad*
where *smack-inode-unlink s dir d* ≡ *do*
ip← *return*(*the*(*d-backing-inode s d*));
ad ← *return* (*SOME x* :: *smk-audit-info .True*);
rc ← *smk-curacc s* (*smk-of-inode s ip*) *MAY-WRITE ad*;
rc ← *return*(*smk-bu-inode s ip MAY-WRITE rc*);
rc ← (*if rc* = *0*
then do
rc ← *smk-curacc s* (*smk-of-inode s dir*) *MAY-WRITE ad*;
return(*smk-bu-inode s dir MAY-WRITE rc*)
od
else return rc);
return(*rc*)
od

definition *smack-inode-rmdir* :: *State'* ⇒ *inode* ⇒*dentry*⇒ (*State'*, *int*) *nondet-monad*
where *smack-inode-rmdir s dir d* ≡ *do*
ip← *return*(*the*(*d-backing-inode s d*));
ad ← *return* (*SOME x* :: *smk-audit-info .True*);
rc ← *smk-curacc s* (*smk-of-inode s ip*) *MAY-WRITE ad*;
rc ← *return*(*smk-bu-inode s ip MAY-WRITE rc*);
rc ← (*if rc* = *0*
then do
rc ← *smk-curacc s* (*smk-of-inode s dir*) *MAY-WRITE ad*;
return(*smk-bu-inode s dir MAY-WRITE rc*)
od
else return rc);
return(*rc*)
od

definition *smack-inode-rename* :: *State'* ⇒ *inode* ⇒*dentry* ⇒ *inode* ⇒ *dentry*⇒
(*State'*, *int*) *nondet-monad*
where *smack-inode-rename s old-inode old-dentry new-indoe new-dentry* ≡ *do*
isp← *return*(*the*(*d-backing-inode s old-dentry*));
ad ← *return* (*SOME x* :: *smk-audit-info .True*);
rc ← *smk-curacc s* (*smk-of-inode s isp*) *MAY-READWRITE ad*;
rc ← *return*(*smk-bu-inode s isp MAY-READWRITE rc*);
rc ← (*if rc* = *0* ∧ *d-is-positive*(*new-dentry*)
then do
rc ← *smk-curacc s* (*smk-of-inode s isp*) *MAY-READWRITE ad*;
return(*smk-bu-inode s* (*the*(*d-backing-inode s new-dentry*))
MAY-READWRITE rc)
od
else return rc);

```
        return(rc)
       od

definition smack-inode-permission :: State' ⇒  inode ⇒int⇒ (State', int) nondet-monad
  where smack-inode-permission s i fmask ≡ do
        sbsp ← (return  (the(sb-security s (i-sb i))));
        no-block ← return(fmask AND MAY-NOT-BLOCK);
        f ← return (fmask AND 15);
        rc← (if f = 0 then
               return 0
             else if ((smk-flags sbsp) AND SMK-SB-UNTRUSTED) ≠ 0 ∧
              (smk-of-inode s i) ≠ (smk-root sbsp) then return (uminus(EACCES))
                    else  if no-block ≠ 0 then return (−ECHILD) else
                 do
                    ad ← return (SOME x :: smk-audit-info .True);
                    mask ← return (nat f);
                    rc ← smk-curacc s (smk-of-inode s i) mask ad;
                    rc ← return(smk-bu-inode s i mask rc ); return rc
                 od
                 );
        return(rc)
       od

definition smack-inode-setattr :: State' ⇒ dentry ⇒iattr ⇒ (State', int) nondet-monad
  where smack-inode-setattr s d attrs ≡ do
        ad ← return (SOME x :: smk-audit-info .True);
        rc← (if ((ia-valid attrs) AND  ATTR-FORCE)  ≠ 0 then
               return 0
             else  do
                    rc ← smk-curacc s (smk-of-inode s (the(d-backing-inode s d)))
MAY-WRITE ad;
                   return(smk-bu-inode s (the(d-backing-inode s d)) MAY-WRITE
rc )
                 od);
        return(rc)
       od

definition smack-inode-getattr :: State' ⇒ path⇒ (State', int) nondet-monad
  where smack-inode-getattr s p ≡ do
        ad ← return (SOME x :: smk-audit-info .True);
        inode ←return (the(d-backing-inode s (p-dentry p)));
        rc ← smk-curacc s (smk-of-inode s inode) MAY-READ ad;
        rc ← return(smk-bu-inode s inode MAY-READ rc );
        return(rc)
       od

definition xattr-ret  :: State' ⇒ dentry ⇒xattr ⇒string⇒ int ⇒ int ⇒ (State',
```

int) *nondet-monad*
  where *xattr-ret s dentry name value size′ flags′* ≡ *do*
      ns ← *return* (*s-user-ns* (*d-sb dentry*));
    rc ← (*if name* = *XATTR-NAME-SMACKTRANSMUTE* ∧ *value* ≠ ′′*true*′′
        then return (−*EINVAL*)
        else
          cap-inode-setxattr s dentry name value size′ flags′
        );
      return(*rc*)
    od

definition *set-check-priv* :: *xattr* ⇒ *int*
  where *set-check-priv name* ≡ *case name of XATTR-NAME-SMACK* ⇒ *1* |
    XATTR-NAME-SMACKIPIN ⇒ *1* |
    XATTR-NAME-SMACKIPOUT ⇒ *1* |
    XATTR-NAME-SMACKEXEC ⇒ *1* |
    XATTR-NAME-SMACKMMAP ⇒ *1* |
    XATTR-NAME-SMACKTRANSMUTE ⇒ *1* |
    - ⇒ *0*

definition *set-check-import* :: *xattr* ⇒ *int*
  where *set-check-import name* ≡ *case name of XATTR-NAME-SMACK* ⇒ *1* |
    XATTR-NAME-SMACKIPIN ⇒ *1* |
    XATTR-NAME-SMACKIPOUT ⇒ *1* |
    XATTR-NAME-SMACKEXEC ⇒ *1* |
    XATTR-NAME-SMACKMMAP ⇒ *1* |
    - ⇒ *0*

definition *set-check-star* :: *xattr* ⇒ *int*
  where *set-check-star name* ≡ *case name of XATTR-NAME-SMACKEXEC* ⇒
1 |
    XATTR-NAME-SMACKMMAP ⇒ *1* |
    - ⇒ *0*

definition *smk-import-entry* :: *State′* ⇒ *string* ⇒ *int*⇒ (*State′*, *smack-known option*) *nondet-monad*
  where *smk-import-entry s str len′* ≡ *do*
      rc ← *return* (*Some*(*SOME x* :: *smack-known* .*True*));

      return(*rc*)
    od

definition *smack-inode-setxattr* :: *State′* ⇒ *dentry* ⇒*xattr* ⇒*string*⇒ *int* ⇒ *int* ⇒ (*State′*, *int*) *nondet-monad*
  where *smack-inode-setxattr s dentry name value size′ flags′* ≡ *do*
      ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);

skp ← *return* (*SOME x* :: *smack-known option*. *True*);
check-priv ← *return* (*set-check-priv name*);
check-import ← *return*(*set-check-import name*);
check-star ← *return* (*set-check-star name*);
rc ← *xattr-ret s dentry name value size′ flags′*;
rc ← (*if* (*rc* = *0*) ∧ *check-import* ≠ *0 then*
do
skp ← (
if size′ > *0 then smk-import-entry s value size′*
else return None
);
if (*skp* = *None*) ∨
(*check-star* ≠ *0* ∧ ((*the*(*skp*) = *smack-known-star*) ∨ (*the*(*skp*) = *smack-known-web*)))
then return (−*EINVAL*)
else return 0
od
else
return rc
);
inode ←*return* (*the*(*d-backing-inode s dentry*));
rc ← *smk-curacc s* (*smk-of-inode s inode*) *MAY-WRITE ad*;
rc ← *return*(*smk-bu-inode s inode MAY-WRITE rc*);
return(*rc*)
od

definition *smack-inode-post-setxattr* :: *State′* ⇒ *dentry* ⇒*xattr* ⇒*string*⇒ *int* ⇒ *int* ⇒ (*State′*, *unit*) *nondet-monad*
where *smack-inode-post-setxattr s dentry name value size′ flags′* ≡ *do*
skp ← *return* (*SOME x* :: *smack-known* .*True*);
inode ←*return* (*the*(*d-backing-inode s dentry*));
isp ← *return* (*the*(*i-security s inode*));
if name = *XATTR-NAME-SMACKTRANSMUTE then*
do
modify(λ*s* .*s*⦇*i-security* := (*i-security s*)
(*inode* := *Some*(*isp*⦇ *smk-iflags* :=
(*bitOR* (*smk-iflags isp*) *SMK-INODE-TRANSMUTE*)⦈))⦈);
return()
od else
case name of XATTR-NAME-SMACK ⇒
do
skp← *smk-import-entry s value size′*;
if skp ≠ *None then*
do modify(λ*s* .*s*⦇*i-security* := (*i-security s*)(*inode* := *Some*(*isp*⦇ *smk-inode* := *the skp*⦈))⦈);
return()
od else return ()

od |
XATTR-NAME-SMACKEXEC ⇒
do
skp← *smk-import-entry s value size'*;
if skp ≠ *None then*
do modify(λ*s* .*s*⦇*i-security* := (*i-security s*)(*inode* := *Some*(*isp*⦇
smk-itask := *skp*⦈))⦈);
return()
od else return ()
od |
XATTR-NAME-SMACKMMAP ⇒
do
skp← *smk-import-entry s value size'*;
if skp ≠ *None then*
do modify(λ*s* .*s*⦇*i-security* := (*i-security s*)(*inode* := *Some*(*isp*⦇
smk-mmap := *skp*⦈))⦈);
return()
od else return ()
od
od

definition *smack-inode-getxattr* :: *State'* ⇒ *dentry* ⇒*xattr*⇒ (*State'*, *int*) *nondet-monad*
where *smack-inode-getxattr s dentry name* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);
inode ←*return* (*the*(*d-backing-inode s dentry*));
rc ← *smk-curacc s* (*smk-of-inode s inode*) *MAY-READ ad*;
rc ← *return*(*smk-bu-inode s inode MAY-READ rc*);
return(*rc*)
od

definition *xatrr-remove* :: *xattr* ⇒ *bool*
where *xatrr-remove name* ≡ *case name of XATTR-NAME-SMACK* ⇒ *True* |
XATTR-NAME-SMACKIPIN ⇒ *True*|
XATTR-NAME-SMACKIPOUT ⇒ *True* |
XATTR-NAME-SMACKEXEC ⇒ *True* |
XATTR-NAME-SMACKMMAP ⇒ *True* |
XATTR-NAME-SMACKTRANSMUTE ⇒
True |
- ⇒ *False*
record *sysConfig* = *CONFIG-USER-NS* :: *bool*

definition *privileged-wrt-inode-uidgid* :: *ns* ⇒ *inode* ⇒ *bool*
where *privileged-wrt-inode-uidgid ns i* ≡ *True*

definition *capable-wrt-inode-uidgid* :: *State'* => *inode* ⇒ *int* ⇒ *bool*
where *capable-wrt-inode-uidgid s i cap* ≡ *let ns* = *user-ns* (*current-cred* (*get-cur-task s*)) *in*
(*ns-capable ns cap*) ∧ *privileged-wrt-inode-uidgid ns i*

definition *cap-inode-removexattr* :: $State' \Rightarrow dentry \Rightarrow xattr \Rightarrow (State', int)$ *nondet-monad*
where *cap-inode-removexattr s dentry name* $\equiv$ *do*
ns $\leftarrow$ *return* (*s-user-ns* (*d-sb dentry*));
rc $\leftarrow$ (*if name* $\neq$ *XATTR-SECURITY-PREFIX then return 0 else*
if name = *XATTR-NAME-CAPS then*
do
inode$\leftarrow$ *return* (*d-backing-inode s dentry*);
if inode = *None then return* ($-$*EINVAL*) *else*
if $\neg$(*capable-wrt-inode-uidgid s* (*the inode*) *CAP-SETFCAP*) *then*
return ($-$*EPERM*)
else
return 0
od
else
if $\neg$(*ns-capable ns CAP-SYS-ADMIN*) *then return* ($-$*EPERM*)
else
return 0
);
return(*rc*)
od

definition *smack-inode-removexattr* :: $State' \Rightarrow dentry \Rightarrow xattr \Rightarrow (State', int)$
nondet-monad
where *smack-inode-removexattr s dentry name* $\equiv$ *do*
ad $\leftarrow$ *return* (*SOME x* :: *smk-audit-info .True*);
rc $\leftarrow$ (*if xatrr-remove name then if* $\neg$(*smack-privileged s CAP-MAC-ADMIN*)
then return ($-$*EPERM*)
else return 0
else cap-inode-removexattr s dentry name);
rc $\leftarrow$ (*if rc* $\neq$ *0 then return rc else*
do
inode $\leftarrow$*return* (*the*(*d-backing-inode s dentry*));
rc $\leftarrow$ *smk-curacc s* (*smk-of-inode s inode*) *MAY-READ ad*;
rc $\leftarrow$ *return*(*smk-bu-inode s inode MAY-READ rc*);
if rc $\neq$ *0 then return rc*
else
do
inode $\leftarrow$ *return*(*the*(*d-backing-inode s dentry*));
isp $\leftarrow$ *return* (*the*(*i-security s inode*));
if name = *XATTR-NAME-SMACK then*
do
sbp $\leftarrow$ *return*(*d-sb dentry*);
sbsp $\leftarrow$ *return*(*the*(*sb-security s sbp*));
modify(λs .*s*(\|*i-security* := (*i-security s*)(*inode* := *Some*(*isp*(\|
smk-inode := *smk-default sbsp*\|)))\|));
return 0
od

else
if name = XATTR-NAME-SMACKEXEC then do
modify(λs .s(|i-security := (i-security s)(inode := Some(isp(|
smk-itask := None|)))|));
return 0
od
else
if name = XATTR-NAME-SMACKMMAP then do
modify(λs .s(|i-security := (i-security s)(inode := Some(isp(|
smk-mmap := None|)))|));
return 0
od
else if name = XATTR-NAME-SMACKTRANSMUTE then do
iflags ← return(smk-iflags isp AND (NOT
SMK-INODE-TRANSMUTE));
modify(λs .s(|i-security := (i-security s)(inode := Some(isp(|
smk-iflags := iflags|)))|));
return 0
od
else return 0

od
od);
return(rc)
od

definition *kstrdup str ≡ if length(str) = 0 then None else Some str*

definition *smack-inode-getsecurity :: State′ ⇒ inode ⇒xattr⇒ Void⇒ bool⇒*
(State′, int) nondet-monad
where *smack-inode-getsecurity s inode name buffer alloc ≡ do*
ad ← return (SOME x :: smk-audit-info .True);
isp ← return (SOME x :: smack-known .True);
ip ← return (inode);
rc ← (if name = XATTR-SMACK-SUFFIX then
do
isp ← return(smk-of-inode s inode);
return (length(smk-known isp))
od
else do
sbp ← return(i-sb ip);
if (s-magic sbp ≠ SOCKFS-MAGIC) then return(−EOPNOTSUPP)
else do
sock ← return (SOCKET-I ip);
ssp ← return (the(sk-security s (the(sk sock))));
rc ←(if name = XATTR-SMACK-IPIN then
do isp← return(smk-in ssp);
if alloc then do buffer ← return(kstrdup (smk-known

isp));
if buffer = *None then return*
(*ENOMEM*)
else return(*int* (*length*(*smk-known*
isp)))
od else return(*int* (*length*(*smk-known*
isp)))
od else
if name = *XATTR-SMACK-IPOUT then*
do isp← *return*(*smk-out ssp*);
if alloc then do buffer ← *return*(*kstrdup* (*smk-known*
isp));
if buffer = *None then return*
(−*ENOMEM*)
else return(*int* (*length*(*smk-known*
isp)))
od else return(*int* (*length*(*smk-known*
isp)))
od
else return (−*EOPNOTSUPP*)
); *return rc*
od

od
);
return(*rc*)
od

term *s*⦇*i-security* := (*i-security s*)(*inode* := *Some*(*nsp*⦇*smk-inode*:= (*the*(*skp*)),
smk-iflags :=(*bitOR* (*smk-iflags nsp*)
SMK-INODE-INSTANT)⦈))⦈

definition *smack-inode-setsecurity* :: *State'* ⇒ *inode* ⇒*xattr*⇒ *Void*⇒ *nat* ⇒ *int*⇒ (*State'*, *int*) *nondet-monad*
where *smack-inode-setsecurity s inode name value size' flg* ≡ *do*
nsp ← *return* (*the* ((*i-security s*) *inode*));
value ← *return*(*SOME x*. *String x* = *value*);
skp ← *return* (*SOME x* :: *smack-known* .*True*);
rc ← (*if length*(*value*) = *0* ∨ *size'* > *SMK-LONGLABEL* ∨ *size'* = *0* *then*

return (−*EINVAL*)
else do
skp ← *smk-import-entry s value size'*;
if skp = *None then return* (−*ENOMEM*) *else*
if (*name* = *XATTR-SMACK-SUFFIX*) *then*
do
modify(λ*s* .*s*⦇*i-security* := (*i-security s*)(*inode* :=
Some(*nsp*⦇*smk-inode*:= (*the*(*skp*)),

smk-iflags :=(*bitOR* (*smk-iflags nsp*)
SMK-INODE-INSTANT)⦈))⦈);
return 0
od

else if (*s-magic* (*i-sb inode*) $\neq$ *SOCKFS-MAGIC*) *then*
return(−*EOPNOTSUPP*)
else do
sock ← *return* (*SOCKET-I inode*);
ssp ← *return* (*the*(*sk-security s* (*the*(*sk sock*))));
rc ←(*if name* = *XATTR-SMACK-IPIN then*
do isp← *return*(*smk-in ssp*);
modify(λ*s* .*s*);
return 0
od else
if name = *XATTR-SMACK-IPOUT then*
do isp← *return*(*smk-out ssp*);
return 0
od
else return (−*EOPNOTSUPP*)
); *return rc*
od
od
);
return(*rc*)
od

definition *smack-inode-listsecurity* :: *State*′ ⇒ *inode* ⇒*Void*⇒*int*⇒ (*State*′, *int*)
nondet-monad
where *smack-inode-listsecurity s inode buffer buffer-size* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);
len ← *return*(*17*);
return(*len*)
od

definition *smack-inode-getsecid* :: *State*′ ⇒ *inode* ⇒*int*⇒ (*State*′, *unit*) *nondet-monad*
where *smack-inode-getsecid s inode secid*′ ≡ *do*
skp ← *return* (*smk-of-inode s inode*);
secid ← *return*(*smk-secid skp*);
return()
od

29.48 file hooks

definition *get-file-name s f* ≡ *SOME n* . *files s n* = *Some f*

type-synonym *smackfile* = *Files*

definition *smack-file-alloc-security* :: *State′* ⇒ *smackfile*⇒ (*State′*, *int*) *nondet-monad*
  where *smack-file-alloc-security s file′* ≡ *do*
  f ← *return* (*smk-of-current s*);
  fsp ← *return* (*f-security s file′*);
  if fsp ≠ *None then return* (−*EEXIST*)
  else do
  modify(λ*s* .*s*(|*f-security* := (*f-security s*)(*file′* := *Some f*)|));
  rc ← *return*(*0*);
  return(*rc*)
  od
  od

definition *smack-file-free-security* :: *State′* ⇒ *smackfile*⇒ (*State′*, *unit*) *nondet-monad*
  where *smack-file-free-security s file′* ≡ *do*
  fsp ← *return* (*f-security s file′*);
  if fsp = *None then return* ()
  else do
  modify(λ*s* .*s*(|*f-security* := (*f-security s*)(*file′* := *None*)|));
  return() *od*
  od

definition *smack-file-ioctl* :: *State′* ⇒ *smackfile* ⇒ *IOC-DIR* ⇒ *nat*⇒ (*State′*, *int*) *nondet-monad*
  where *smack-file-ioctl s file′ cmd arg* ≡ *do*
  ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);
  inode ← *return*(*file-inode file′*);

  rc←(*if unlikely*(*IS-PRIVATE*(*inode*)) *then return 0 else*
  do
  rc ← (*case cmd of IOC-WRITE* ⇒
  do
  rc ← *smk-curacc s* (*smk-of-inode s inode*) *MAY-WRITE*
ad;
  return(*smk-bu-file s file′ MAY-WRITE rc*)
  od |
  IOC-READ ⇒
  do
  rc ← *smk-curacc s* (*smk-of-inode s inode*) *MAY-READ*
ad;
  return(*smk-bu-file s file′ MAY-READ rc*)
  od | - ⇒ *return 0*);
  return rc
  od);
  return(*rc*)
  od

definition *smack-file-lock* :: *State'* ⇒ *smackfile* ⇒ *nat* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-file-lock s file' cmd* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info .True*);
inode← *return*(*file-inode*(*file'*));
rc←(*if unlikely*(*IS-PRIVATE*(*inode*)) *then return 0 else*
do
rc ← *smk-curacc s* (*smk-of-inode s inode*) *MAY-LOCK ad*;
return(*smk-bu-file s file' MAY-LOCK rc*)
od);
return(*rc*)
od

definition *smack-file-fcntl* :: *State'* ⇒ *smackfile* ⇒ *nat* ⇒*nat*⇒ (*State'*, *int*) *nondet-monad*
where *smack-file-fcntl s file' cmd arg* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info .True*);
inode← *return*(*file-inode*(*file'*));
rc←(*if unlikely*(*IS-PRIVATE*(*inode*)) *then return 0 else*
if cmd =*F-SETLK* ∨ *cmd* = *F-SETLKW then*
do
rc ← *smk-curacc s* (*smk-of-inode s inode*) *MAY-LOCK ad*;
return(*smk-bu-file s file' MAY-LOCK rc*)
od
else if cmd = *F-SETOWN* ∨ *cmd* = *F-SETSIG then*
do
rc ← *smk-curacc s* (*smk-of-inode s inode*) *MAY-WRITE ad*;
return(*smk-bu-file s file' MAY-WRITE rc*)
od
else
return 0
);
return(*rc*)
od

definition *smack-mmap-file* :: *State'* ⇒ *smackfile option* ⇒ *nat* ⇒*nat* ⇒ *nat*⇒ (*State'*, *int*) *nondet-monad*
where *smack-mmap-file s file' reqprot prot flags'* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info .True*);
rc ← (*if file'* = *None* ∨ (*unlikely*(*IS-PRIVATE*(*file-inode*(*the*(*file'*)))))
then return 0
else do
isp ← *return*(*the*(*i-security s* (*file-inode*(*the*(*file'*)))));
if smk-mmap isp = *None then return 0 else*
do
sbsp ← *return*(*the*((*sb-security s*) (*i-sb* (*file-inode*(*the* (*file'*))))));
if (*smk-flags sbsp AND SMK-SB-UNTRUSTED*) ≠ *0* ∧ (*the*(*smk-mmap isp*) ≠ *smk-root sbsp*)
then return (−*EACCES*)

else do
mkp ← *return*(*the*(*smk-mmap isp*));
tsp ← *return*(*current-security s*);
skp ← *return*(*smk-of-current s*);
return 0
od
od
od
);
return(*rc*)
od

consts *dac-mmap-min-addr* :: *nat*
consts *init-user-ns* :: *ns*

definition *cap-capable-boby* :: *State′* ⇒ *Cred* ⇒ *ns* ⇒*int* ⇒*int*⇒ (*State′*, *int*) *nondet-monad*
where *cap-capable-boby s c ns cap audit* ≡ *do*
rc ←(*if ns* = *user-ns c then*
if (*cap-raised* (*cap-effective c*) *cap*) ≠ *0 then return 0 else return* (−*EPERM*)
else
if ns-level ns ≤ *ns-level* (*user-ns c*) *then return* (−*EPERM*)
else if uid-eq (*owner ns*) (*euid c*)
then return 0 else return (−*EPERM*)
);
return rc
od

definition *cap-capable* :: *State′* ⇒ *Cred* ⇒ *ns* ⇒*int* ⇒*int*⇒ (*State′*, *int*) *nondet-monad*
where *cap-capable s c targ-ns cap audit* ≡ *do*
ns ← *return*(*targ-ns*);
return(*0*)
od

definition *cap-mmap-addr* :: *State′* ⇒ *nat*⇒ (*State′*, *int*) *nondet-monad*
where *cap-mmap-addr s addr* ≡ *do*
ret← *return* (*0*);
ret ← (*if addr* < *dac-mmap-min-addr then do*
ret ← *cap-capable s* (*current-cred* (*get-cur-task s*)) *init-user-ns CAP-SYS-RAWIO SECURITY-CAP-AUDIT*;
return ret
od
else return ret);
return(*ret*)
od

definition *smack-file-set-fowner* :: *State'* ⇒ *smackfile*⇒ (*State'*, *unit*) *nondet-monad*
where *smack-file-set-fowner s file'* ≡ *do*
f ← *return* (*smk-of-current s*);
modify(λ*s* .*s*⦇*f-security* := (*f-security s*)(*file'* := *Some f*)⦈);
return()
od

definition *container-of-smack* :: *fown-struct* ⇒ *smackfile*
where *container-of-smack fown* ≡ *SOME f* . *fown* = *f-owner f*

definition *smack-file-send-sigiotask* :: *State'* ⇒ *Task* ⇒ *fown-struct* ⇒*int*⇒ (*State'*, *int*) *nondet-monad*
where *smack-file-send-sigiotask s tsk' fown signum* ≡ *do*
skp ← *return* (*SOME x* :: *smack-known* .*True*);
tkp ← *return* (*smk-of-task* (*the*((*t-security s*) (*current-cred tsk'*))));
file' ← *return* (*container-of-smack fown*);
skp ← *return*(*the*(*f-security s file'*));
rc ← *smk-access s skp tkp MAY-DELIVER None*;
rc ← *return*(*smk-bu-note s ''sigiotask'' skp tkp MAY-DELIVER rc*);
tcred ← *return*(*task-cred*(*tsk'*));
rc ← (*if rc* ≠ *0* ∧ (*smack-privileged-cred CAP-MAC-OVERRIDE tcred*)
then return 0
else return rc);
return(*rc*)
od

definition *smack-file-receive* :: *State'* ⇒ *smackfile*⇒ (*State'*, *int*) *nondet-monad*
where *smack-file-receive s file'* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);
may ← *return 0*;
inode← *return*(*file-inode*(*file'*));
rc←(*if unlikely*(*IS-PRIVATE*(*inode*)) *then return 0 else*
do
rc ← (*if* (*s-magic* (*i-sb inode*)) = *nat SOCKFS-MAGIC then*
do
sock ← *return*(*SOCKET-I inode*);
ssp ← *return*(*the*(*sk-security s* (*the*(*sk sock*))));
tsp ← *return*(*current-security s*);
rc ← *smk-access s* (*smk-task tsp*) (*smk-out ssp*) *MAY-WRITE* (*Some ad*) ;
rc ← *return*(*smk-bu-file s file' may rc*);
rc ← (*if rc* < *0 then return rc*
else
do
rc ← *smk-access s* (*smk-in ssp*) (*smk-task tsp*) *MAY-WRITE* (*Some ad*) ;
rc ← *return*(*smk-bu-file s file' may rc*); *return rc*
od
);

return rc
od else if (*f-mode file′ AND FMODE-READ*) ≠ *0 then*
do
may ← *return* (*MAY-READ*);
rc ← *smk-curacc s* (*smk-of-inode s inode*) *may ad*;
rc ← *return*(*smk-bu-file s file′ MAY-LOCK rc*);
return rc

od else if(*f-mode file′ AND FMODE-WRITE*) ≠ *0 then*
do
may ← *return* (*bitOR may MAY-READ*);
rc ← *smk-curacc s* (*smk-of-inode s inode*) *may ad*;
rc ← *return*(*smk-bu-file s file′ MAY-LOCK rc*);
return rc

od else
do
rc ← *smk-curacc s* (*smk-of-inode s inode*) *may ad*;
rc ← *return*(*smk-bu-file s file′ MAY-LOCK rc*);
return rc
od
);

return(*rc*)
od);
return(*rc*)
od

definition *smack-file-open* :: *State′* ⇒ *smackfile*⇒ (*State′*, *int*) *nondet-monad*
where *smack-file-open s file′* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);
inode← *return*(*file-inode*(*file′*));
tsp ← *return*(*the*(*t-security s* (*f-cred file′*)));
rc←(
do
rc ← *smk-tskacc s tsp* (*smk-of-inode s inode*) *MAY-READ ad*;
return(*smk-bu-credfile s* (*f-cred file′*) *file′ MAY-READ rc*)
od);
return(*rc*)
od

29.49 task hooks

definition *smack-cred-alloc-blank* :: *State′* ⇒ *Cred* ⇒ *nat* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-cred-alloc-blank s cred′ gfp′* ≡ *do*
tsp ← *return* (*SOME x* :: *task-smack* .*True*);
t← *return* (*SOME x* :: *smack-known* .*True*);
tsp← *return*(*new-task-smack t t gfp′*);

```
        rc←( if tsp = None then return (−ENOMEM)
              else
              do
               modify(λs .s⦇t-security := (t-security s)(cred′ := tsp )⦈);
               return 0
              od
              );
        return(rc)
       od
```

definition *smack-cred-free* :: *State′* ⇒ *Cred* ⇒(*State′*, *unit*) *nondet-monad*

```
  where smack-cred-free s cred′ ≡ do
        tsp ← return (SOME x :: task-smack .True);
         modify(λs .s⦇t-security := (t-security s)(cred′ := None )⦈);
         return()
       od
```

definition *smack-cred-prepare* :: *State′* ⇒ *Cred* ⇒*Cred* ⇒ *nat*⇒(*State′*, *int*) *nondet-monad*

```
  where smack-cred-prepare s new old g ≡ do
        old-tsp ← return (the ((t-security s) old));
        new-tsp ←return (SOME x :: task-smack .True);
         new-tsp ← return(new-task-smack (smk-task old-tsp) (smk-task old-tsp) g
);
        rc← (if new-tsp = None then return (−ENOMEM)
             else do
                  new-tsp′ ← return(the(new-task-smack (smk-task old-tsp) (smk-task
old-tsp) g));
                      modify(λs .s⦇t-security := (t-security s)(new := new-tsp )⦈);
                     rc← (smk-copy-rules s (smk-rule new-tsp′) (smk-rule old-tsp) g );
                      rc←( if rc ≠ 0 then return rc else
                      do
                             rc← (smk-copy-relabel s (smk-relabel new-tsp′) (smk-relabel
old-tsp) g);
                         if rc ≠ 0 then return rc
                         else
                           return 0
                      od);
                      return rc
                    od
  );
         return(rc)
       od
```

definition *smack-cred-getsecid* :: *State′* ⇒ *Cred* ⇒*u32*⇒(*State′*, *unit*) *nondet-monad*

```
  where smack-cred-getsecid s c seci ≡ do
         skp ← return (SOME x:: smack-known. True);
```

skp← *return* (*smk-of-task* (*the* (*t-security s c*)));
seci ← *return*(*smk-secid skp*);
return()
od

definition *smack-kernel-act-as* :: *State'* ⇒ *Cred* ⇒*u32*⇒(*State'*, *int*) *nondet-monad*
where *smack-kernel-act-as s cred' seci* ≡ *do*
new-tsp ← *return* (*the*(*t-security s cred'*));
i← *smack-from-secid seci*;
modify(λ*s* .*s*(|*t-security* := (*t-security s*)(*cred'* := *Some* (*new-tsp*(|*smk-task* := *the i*|)))|));
return(*0*)
od

definition *smack-kernel-create-files-as* :: *State'* ⇒ *Cred* ⇒ *inode*⇒(*State'*, *int*) *nondet-monad*
where *smack-kernel-create-files-as s new inode'* ≡ *do*
isp← *return* (*the*(*i-security s inode'*));
tsp ← *return* (*the*(*t-security s new*));
modify(λ*s* .*s*(|*t-security* := (*t-security s*)(*new* := *Some* (*tsp*(|*smk-forked* := *smk-inode isp*,*smk-task* := *smk-forked tsp*|)))|));
return(*0*)
od

definition *smack-cred-transfer* :: *State'* ⇒ *Cred* ⇒*Cred*⇒(*State'*, *unit*) *nondet-monad*
where *smack-cred-transfer s new old* ≡ *do*
old-tsp← *return* (*the*(*t-security s old*));
new-tsp ← *return* (*the*(*t-security s new*));
modify(λ*s* .*s*(|*t-security* := (*t-security s*)
(*new* := *Some* (*new-tsp*(|*smk-forked* := *smk-task old-tsp*,
smk-task :=*smk-task old-tsp* |)))|));
return()
od

definition *smk-curacc-on-task* :: *State'* ⇒ *Task* ⇒ *int* ⇒ *string*⇒ (*State'*, *int*) *nondet-monad*
where *smk-curacc-on-task s p access' caller'* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);
skp← *return*(*smk-of-task-struct s p*);
rc←(*do*
rc ← *smk-curacc s skp access' ad*;
return(*smk-bu-task s p access' rc*)
od);
return(*rc*)

od

definition *smack-task-setpgid* :: *State'* ⇒ *Task* ⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-task-setpgid s p pgid* ≡ *do*
rc ← *smk-curacc-on-task s p MAY-WRITE ''smack-task-setpgid''*;
return(*rc*) *od*

definition *smack-task-getpgid* :: *State'* ⇒ *Task* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-task-getpgid s p* ≡ *do*
rc ← *smk-curacc-on-task s p MAY-READ ''smack-task-getpgid''*;
return(*rc*) *od*

definition *smack-task-getsid* :: *State'* ⇒ *Task* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-task-getsid s p* ≡ *do*
rc ← *smk-curacc-on-task s p MAY-READ ''smack-task-getsid''*;
return(*rc*) *od*

definition *smack-task-getsecid* :: *State'* ⇒ *Task* ⇒ *nat* ⇒ (*State'*, *unit*) *nondet-monad*
where *smack-task-getsecid s p secid'* ≡ *do*
skp← *return*(*smk-of-task-struct s p*);
secid'← *return*(*smk-secid skp*);
return() *od*

definition *smack-task-setnice* :: *State'* ⇒ *Task* ⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-task-setnice s p nice* ≡ *do*
rc ← *smk-curacc-on-task s p MAY-WRITE ''smack-task-setnice''*;
return(*rc*) *od*

definition *smack-task-setioprio* :: *State'* ⇒ *Task* ⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-task-setioprio s p ioprio* ≡ *do*
rc ← *smk-curacc-on-task s p MAY-WRITE ''smack-task-setioprio''*;
return(*rc*) *od*

definition *smack-task-getioprio* :: *State'* ⇒ *Task* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-task-getioprio s p* ≡ *do*
rc ← *smk-curacc-on-task s p MAY-READ ''smack-task-getioprio''*;
return(*rc*) *od*

definition *smack-task-setscheduler* :: *State'* ⇒ *Task* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-task-setscheduler s p* ≡ *do*
rc ← *smk-curacc-on-task s p MAY-WRITE ''smack-task-setscheduler''*;

return(*rc*)
od

definition *smack-task-getscheduler* :: *State′* ⇒ *Task* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-task-getscheduler s p* ≡ *do*
rc ← *smk-curacc-on-task s p MAY-READ ′′smack-task-setscheduler′′*;
return(*rc*)
od

definition *smack-task-movememory* :: *State′* ⇒ *Task* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-task-movememory s p* ≡ *do*
rc ← *smk-curacc-on-task s p MAY-WRITE ′′smack-task-movememory′′*;
return(*rc*)
od

definition *smack-task-kill* :: *State′* ⇒ *Task* ⇒*siginfo* ⇒ *int* ⇒ *Cred option*⇒ (*State′*, *int*) *nondet-monad*
where *smack-task-kill s p info sig cred′* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);
skp← *return* (*SOME x* :: *smack-known* .*True*);
tkp ← *return*(*smk-of-task-struct s p*);
rc ←(*if sig* = *0 then return 0 else if cred′* = *None then*
do rc ←(*smk-curacc s tkp MAY-DELIVER ad*);
return(*smk-bu-task s p MAY-DELIVER rc*) *od*
else do
ad ← *return* (*Some ad*);
skp ← *return* (*smk-of-task* (*current-security s*));
rc ← *smk-access s skp tkp MAY-DELIVER ad*;
rc ← *return* (*smk-bu-note s ′′USB signal′′ skp tkp MAY-DELIVER rc*) ;
return rc
od
);
return(*rc*)
od

definition *smack-task-to-inode* :: *State′* ⇒ *Task* ⇒ *inode* ⇒ (*State′*, *unit*) *nondet-monad*
where *smack-task-to-inode s p i* ≡ *do*
isp ← *return* (*the*(*i-security s i*));
skp ← *return*(*smk-of-task-struct s p*);
f ← *return*(*bitOR* (*smk-iflags isp*) *SMK-INODE-INSTANT*);
modify(*λs* .*s*⦇*i-security* := (*i-security s*) (*i* := *Some* (*isp*⦇*smk-inode* := *skp*, *smk-iflags* := *f*⦈))⦈);
return()*od*

definition *prepare-creds* :: *State'* ⇒ *State'* × *Cred option*
where*prepare-creds s* ≡ *let task* = *get-cur-task s*;
new = *SOME x*:: *Cred*. *True*;
old = *cred task in*
if fst(*the-run-state*((*smack-cred-prepare s new old 0*)) *s*) < *0*
then (*s*,*None*)
else (*s*,*Some new*)

29.50 kern$_{i}pc_{p}erm$

definition *get-msg-id* :: *State'* ⇒ *msg-msg*⇒ *int*
where *get-msg-id s msg* ≡ *SOME id* . (*msg-msgs s*) *id* = *Some msg*

definition *get-msg-queue-id*:: *State'* ⇒ *kern-ipc-perm* ⇒ *int*
where *get-msg-queue-id s msg* ≡ *SOME id* . (*msg-queues s*) *id* = *Some msg*

definition *smack-flags-to-may* :: *int* ⇒ *int*
where *smack-flags-to-may flag* ≡ *let may* = *0 in*
if (*flag AND S-IRUGO*) ≠ *0*
then (*bitOR may MAY-READ*)
else
if (*flag AND S-IWUGO*) ≠ *0*
then (*bitOR may MAY-WRITE*)
else
if (*flag AND S-IXUGO*) ≠ *0*
then (*bitOR may MAY-EXEC*)
else may

definition *get-ipc-security* :: *State'* ⇒ *kern-ipc-perm* ⇒ *smack-known*
where *get-ipc-security s ipc'* ≡ (*the*((*ipc-security s*) *ipc'*))

definition *smack-ipc-permission* :: *State'* ⇒ *kern-ipc-perm* ⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-ipc-permission s ipp flag* ≡ *do*
ad ← *return* (*SOME x* :: *smk-audit-info* .*True*);
iskp ← *return* (*get-ipc-security s ipp*);
may ← *return*(*smack-flags-to-may flag*);
rc ← *smk-curacc s iskp may ad*;
rc← *return*(*smk-bu-current s ''svipc'' iskp may rc*);
return(*rc*)
od

definition *smack-ipc-getsecid* :: *State'* ⇒ *kern-ipc-perm* ⇒ *nat* ⇒ (*State'*, *unit*) *nondet-monad*

where *smack-ipc-getsecid s ipp flag* $\equiv$ *do*
iskp $\leftarrow$ *return* (*get-ipc-security s ipp*);
secid $\leftarrow$ *return*(*smk-secid iskp*);
return() *od*

definition *smack-msg-msg-alloc-security* :: *State'* $\Rightarrow$ *msg-msg* $\Rightarrow$ (*State'*, *int*) *nondet-monad*
where *smack-msg-msg-alloc-security s msg* $\equiv$ *do*
skp $\leftarrow$ *return* (*smk-of-current s*);
msgs $\leftarrow$ *return* (*msg-security s msg*);
if msgs $\neq$ *None then return*($-$*EEXIST*)
else do
modify(λ*s* .*s*(|*msg-security* := (*msg-security s*)(*msg* := *Some skp*)|));
return(*0*)
od
od

definition *smack-msg-msg-free-security* :: *State'* $\Rightarrow$ *msg-msg* $\Rightarrow$ (*State'*, *unit*) *nondet-monad*
where *smack-msg-msg-free-security s msg* $\equiv$ *do*
msgs $\leftarrow$ *return* (*msg-security s msg*);
if msgs = *None then return* ()
else do
modify(λ*s* .*s*(|*msg-security* := (*msg-security s*)(*msg* := *None*)|));
return()
od
od

definition *smack-of-ipc* :: *State'*$\Rightarrow$*kern-ipc-perm* => *smack-known*
where *smack-of-ipc s isp* $\equiv$ *get-ipc-security s isp*

definition *smack-ipc-alloc-security* :: *State'* $\Rightarrow$ *kern-ipc-perm* $\Rightarrow$ (*State'*, *int*) *nondet-monad*
where *smack-ipc-alloc-security s isp* $\equiv$ *do*
skp $\leftarrow$ *return* (*smk-of-current s*);
modify(λ*s* .*s*(|*ipc-security* := (*ipc-security s*)(*isp*:= *Some skp*)|));
return(*0*)
od

definition *smack-ipc-free-security* :: *State'* $\Rightarrow$ *kern-ipc-perm* $\Rightarrow$ (*State'*, *unit*) *nondet-monad*
where *smack-ipc-free-security s isp* $\equiv$ *do*
modify(λ*s* .*s*(|*ipc-security* := (*ipc-security s*)(*isp*:= *None*)|));
return()
od

definition *smk-curacc-shm* :: *State'* ⇒ *kern-ipc-perm* ⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where *smk-curacc-shm s isp access* ≡ *do*
ssp ← *return*(*smack-of-ipc s isp*);
ad ← *return*(*SOME x* :: *smk-audit-info* .*True*);
rc ← *smk-curacc s ssp access ad* ;
rc ← *return*(*smk-bu-current s ''shm'' ssp access rc*);
return rc
od

definition *smack-shm-associate* :: *State'* ⇒ *kern-ipc-perm* ⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-shm-associate s isp shmflg* ≡ *do*
may ← *return*(*smack-flags-to-may shmflg*);
rc ← *smk-curacc-shm s isp may*;
return rc
od

definition *smack-shm-shmctl* :: *State'* ⇒ *kern-ipc-perm* ⇒ *IPC-CMD* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-shm-shmctl s isp cmd* ≡
do
rc ← (*case cmd of SHM-STAT* => *smk-curacc-shm s isp MAY-READ* |
SHM-STAT-ANY ⇒ *smk-curacc-shm s isp MAY-READ* |

IPC-STAT ⇒ *smk-curacc-shm s isp MAY-READ* |
SHM-LOCK ⇒ *smk-curacc-shm s isp MAY-READWRITE* |
SHM-UNLOCK ⇒ *smk-curacc-shm s isp MAY-READWRITE* |
IPC-SET ⇒ *smk-curacc-shm s isp MAY-READWRITE* |
IPC-RMID ⇒ *smk-curacc-shm s isp MAY-READWRITE* |
IPC-INFO ⇒ *return 0* |
MSG-INFO ⇒ *return 0* |
- ⇒ *return* (−*EINVAL*)
);
return rc
od

definition *smack-shm-shmat* :: *State'* ⇒ *kern-ipc-perm* ⇒*string*⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-shm-shmat s ipc' shmaddr shmflg* ≡ *do*
may ← *return*(*smack-flags-to-may shmflg*);
rc ← *smk-curacc-shm s ipc' may*;
return rc

od

definition *smk-curacc-sem* :: *State'* ⇒ *kern-ipc-perm* ⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where *smk-curacc-sem s isp access* ≡ *do*
ssp ← *return*(*smack-of-ipc s isp*);
ad ← *return*(*SOME x* :: *smk-audit-info* . *True*);
rc ← *smk-curacc s ssp access ad* ;
rc ← *return*(*smk-bu-current s ''sem'' ssp access rc*);
return rc
od

definition *smack-sem-associate* :: *State'* ⇒ *kern-ipc-perm* ⇒ *int* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-sem-associate s isp shmflg* ≡ *do*
may ← *return*(*smack-flags-to-may shmflg*);
rc ← *smk-curacc-sem s isp may*;
return rc
od

definition *smack-sem-semctl* :: *State'* ⇒ *kern-ipc-perm* ⇒ *IPC-CMD* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-sem-semctl s isp cmd* ≡
do
rc ← (*case cmd of GETPID => smk-curacc-sem s isp MAY-READ* |
GETNCNT ⇒ *smk-curacc-sem s isp MAY-READ* |
GETZCNT ⇒ *smk-curacc-sem s isp MAY-READ* |
GETVAL => smk-curacc-sem s isp MAY-READ |
GETALL ⇒ *smk-curacc-sem s isp MAY-READ* |
SEM-STAT ⇒ *smk-curacc-sem s isp MAY-READ* |
SEM-STAT-ANY ⇒ *smk-curacc-sem s isp MAY-READ* |
IPC-STAT ⇒ *smk-curacc-sem s isp MAY-READ* |
SETVAL ⇒ *smk-curacc-sem s isp MAY-READWRITE* |
SETALL ⇒ *smk-curacc-sem s isp MAY-READWRITE* |
IPC-SET ⇒ *smk-curacc-sem s isp MAY-READWRITE* |
IPC-RMID ⇒ *smk-curacc-sem s isp MAY-READWRITE* |
IPC-INFO ⇒ *return 0* |
MSG-INFO ⇒ *return 0* |
\- ⇒ *return* (−*EINVAL*)
);
return rc
od

definition *smack-sem-semop* :: *State′* ⇒ *kern-ipc-perm* ⇒ *sembuf* ⇒ *nat* ⇒*int* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-sem-semop s isp sops nsops alter* ≡ *do*
rc ← *smk-curacc-sem s isp MAY-READWRITE*;
return rc
od

definition *smk-curacc-msq* :: *State′* ⇒ *kern-ipc-perm* ⇒ *int* ⇒ (*State′*, *int*) *nondet-monad*
where *smk-curacc-msq s isp access* ≡ *do*
msp ← *return*(*smack-of-ipc s isp*);
ad ← *return*(*SOME x* :: *smk-audit-info* .*True*);
rc ← *smk-curacc s msp access ad* ;
rc ← *return*(*smk-bu-current s ″msq″ msp access rc*);
return rc
od

definition *smack-msg-queue-associate* :: *State′* ⇒ *kern-ipc-perm* ⇒ *int* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-msg-queue-associate s isp msqflg* ≡ *do*
may ← *return*(*smack-flags-to-may msqflg*);
rc ← *smk-curacc-msq s isp may*;
return rc
od

definition *smack-msg-queue-msgctl* :: *State′* ⇒ *kern-ipc-perm* ⇒ *IPC-CMD* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-msg-queue-msgctl s isp cmd* ≡
do
rc ← (*case cmd of IPC-STAT* => *smk-curacc-msq s isp MAY-READ* |
MSG-STAT ⇒ *smk-curacc-msq s isp MAY-READ* |
MSG-STAT-ANY ⇒ *smk-curacc-msq s isp MAY-READ* |
IPC-SET ⇒ *smk-curacc-msq s isp MAY-READWRITE* |
IPC-RMID ⇒ *smk-curacc-msq s isp MAY-READWRITE* |
IPC-INFO ⇒ *return 0* |
MSG-INFO ⇒ *return 0* |
- ⇒ *return* (−*EINVAL*)
);
return rc
od

definition *smack-msg-queue-msgsnd* :: *State′* ⇒ *kern-ipc-perm* ⇒ *msg-msg* ⇒ *int* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-msg-queue-msgsnd s isp msg msqflg* ≡ *do*
may ← *return*(*smack-flags-to-may msqflg*);

$rc \leftarrow smk\text{-}curacc\text{-}msq\ s\ isp\ may;$
$return\ rc$
od

definition $smack\text{-}msg\text{-}queue\text{-}msgrcv :: State' \Rightarrow\ kern\text{-}ipc\text{-}perm \Rightarrow\ msg\text{-}msg \Rightarrow Task {\Rightarrow} int {\Rightarrow}\ int \Rightarrow (State', int)\ nondet\text{-}monad$
where $smack\text{-}msg\text{-}queue\text{-}msgrcv\ s\ isp\ msg\ p\ long\ msqflg \equiv do$
$rc \leftarrow smk\text{-}curacc\text{-}msq\ s\ isp\ MAY\text{-}READWRITE;$
$return\ rc$
od

29.51 key

definition $get\text{-}key\text{-}id :: State' \Rightarrow\ key {\Rightarrow}\ int$
where $get\text{-}key\text{-}id\ s\ k \equiv SOME\ id\ .\ (keys\ s)\ id = Some\ k$

definition $smack\text{-}key\text{-}alloc :: State' \Rightarrow\ key \Rightarrow Cred \Rightarrow nat \Rightarrow (State', int)\ nondet\text{-}monad$
where $smack\text{-}key\text{-}alloc\ s\ k\ c\ flg \equiv do$
$skp \leftarrow return(\ smk\text{-}of\text{-}task\ (the((t\text{-}security\ s)\ c)\ ));$
$modify(\lambda s\ .s(|key\text{-}security := (key\text{-}security\ s)(k :=\ Some\ skp$
$)|));$
$return\ 0$
od

definition $smack\text{-}key\text{-}free :: State' \Rightarrow\ key\ \Rightarrow (State', unit)\ nondet\text{-}monad$
where $smack\text{-}key\text{-}free\ s\ k\ \equiv do$

$modify(\lambda s\ .s(|key\text{-}security := (key\text{-}security\ s)(k :=\ None\ )|));$
$return\ ()$
od
definition $KEY\text{-}NEED\text{-}ALL \equiv 63$

definition $key\text{-}ref\text{-}to\text{-}ptr :: State' \Rightarrow key\text{-}ref\text{-}t => key\ option$
where $key\text{-}ref\text{-}to\text{-}ptr\ s\ key\text{-}ref \equiv (keys\ s)\ key\text{-}ref$

definition $smack\text{-}key\text{-}permission :: State' \Rightarrow key\text{-}ref\text{-}t {\Rightarrow}\ Cred \Rightarrow nat \Rightarrow (State', int)\ nondet\text{-}monad$
where $smack\text{-}key\text{-}permission\ s\ key\text{-}ref\ c\ perm\ \equiv do$
$tkp \leftarrow return(\ Some(smk\text{-}of\text{-}task\ (the((t\text{-}security\ s)\ c)\ )));$
$ad \leftarrow return(\ SOME\ x :: smk\text{-}audit\text{-}info\ .True);$
$request \leftarrow return\ 0;$
$rc \leftarrow (if\ (int\ perm)\ AND\ (NOT\ KEY\text{-}NEED\text{-}ALL) \neq 0$
$then\ return\ (-EINVAL)$
$else$
do
$keyp \leftarrow return(key\text{-}ref\text{-}to\text{-}ptr\ s\ key\text{-}ref);$
$rc \leftarrow (if\ keyp = None\ then\ return\ (-EINVAL)$

```
                                else if (key-security s)(the keyp) = None then
                                      return 0
                                     else if tkp = None then
                                          return (−EACCES)
                                          else
                                     if smack-privileged-cred CAP-MAC-OVERRIDE
c then
                                             return 0 else
                                             do
                                             request ← (if ((int perm) AND 11) ≠ 0
                                                           then return (bitOR request
MAY-READ )
                                                      else if ((int perm) AND 30) ≠ 0
                                                                   then  return (bitOR
request MAY-WRITE )
                                                           else
                                                              return 0
                                                          );
                                                      rc ← smk-access s (the tkp) (the
((key-security s)(the keyp))) request (Some ad);
                                              rc ← return(smk-bu-note s ''key access''
(the tkp) (the ((key-security s)(the keyp))) request rc);
                                                 return rc
                                             od
                                       );
                                return rc
                             od
                          );

                        return rc
                        od
```

```
definition smack-key-getsecurity :: State′ ⇒  key ⇒ string⇒ (State′, int) nondet-monad
  where smack-key-getsecurity s k  buffer≡ do
                        skp ← return(key-security s k);
                        rc ← (if skp = None then return 0 else
                                 do
                                    skp ← return (the skp);
                                    copy ← return (kstrdup (smk-known skp));
                                        if copy = None then return (−ENOMEM) else
return (length(the(copy))+1)
                                 od
                            );
                        return rc
                        od
```

29.52 sock

type-synonym *smacksock* = *sock*
type-synonym *smacksocket* = *socket*
definition *get-socket-id* :: *State′* ⇒ *smacksocket*⇒ *int*
  where *get-socket-id s sock* ≡ *SOME id* . (*sockets s*) *id* = *Some sock*

definition *smack-unix-stream-connect* :: *State′* ⇒ *smacksock* ⇒ *smacksock* ⇒ *smacksock*⇒ (*State′*, *int*) *nondet-monad*
  where *smack-unix-stream-connect s sock other newsk*≡ *do*

```
            ssp ← return(the(sk-security s sock));
            osp ← return(the(sk-security s other));
            nsp ← return(the(sk-security s newsk));
            ad ← return( SOME x :: smk-audit-info .True);
            ad ← return(Some ad);
           rc ← (if ¬(smack-privileged s CAP-MAC-OVERRIDE) then
                  do
                    skp ← return (smk-out ssp);
                    okp ← return (smk-in osp);
                    rc ← smk-access s skp okp MAY-WRITE ad;
                   rc ← return(smk-bu-note s ''UDS connect'' skp okp
MAY-WRITE rc);
                    if rc = 0 then
                    do
                        okp ← return (smk-out osp);
                        skp ← return (smk-in ssp);
                        rc ← smk-access s okp skp MAY-WRITE ad;
                       rc ← return(smk-bu-note s ''UDS connect'' okp
skp MAY-WRITE rc);
                        return rc
                    od else
                   return rc
                  od
                else
                  return 0
              );
            return rc
            od
```

definition *smack-unix-may-send* :: *State′* ⇒ *socket* ⇒ *socket* ⇒ (*State′*, *int*) *nondet-monad*
  where *smack-unix-may-send s sock other* ≡ *do*

```
            ssp ← return(the(sk-security s (the(sk sock))));
            osp ← return(the(sk-security s (the(sk other))));
            ad ← return( SOME x :: smk-audit-info .True);
            ad ← return(Some ad);
             rc ← (if (smack-privileged s CAP-MAC-OVERRIDE) then
return 0 else
                  do
```

$skp \leftarrow return\ (smk\text{-}out\ ssp);$
$okp \leftarrow return\ (smk\text{-}in\ osp);$
$rc \leftarrow smk\text{-}access\ s\ skp\ okp\ MAY\text{-}WRITE\ ad;$
$rc \leftarrow return(smk\text{-}bu\text{-}note\ s\ ''UDS\ send''\ skp\ okp$
$MAY\text{-}WRITE\ rc);$
$return\ rc$
od
$);$
$return\ rc$
od

definition $netlbl\text{-}sock\text{-}setattr :: sock \Rightarrow Sk\text{-}Family \Rightarrow netlbl\text{-}lsm\text{-}secattr \Rightarrow int$
where $netlbl\text{-}sock\text{-}setattr\ sk'\ family\ secattr \equiv -ENOSYS$

definition $smack\text{-}netlabel :: State' \Rightarrow sock \Rightarrow int \Rightarrow (State', int)\ nondet\text{-}monad$
where $smack\text{-}netlabel\ s\ sock\ labeled \equiv do$
$ssp \leftarrow return(the(sk\text{-}security\ s\ sock));$
$skp \leftarrow return(\ SOME\ x :: smack\text{-}known\ .True);$
$ad \leftarrow return(\ SOME\ x :: smk\text{-}audit\text{-}info\ .True);$
$ad \leftarrow return(Some\ ad);$
$rc \leftarrow (if\ (smk\text{-}out\ ssp =\ (smack\text{-}net\text{-}ambient\ shared)$
$\vee\ labeled = SMACK\text{-}UNLABELED\text{-}SOCKET$
$)$
$then\ return\ 0\ else$
do
$skp \leftarrow return\ (smk\text{-}out\ ssp);$
$rc \leftarrow return(netlbl\text{-}sock\text{-}setattr\ sock\ (sk\text{-}family\ sock)$
$(smk\text{-}netlabel\ skp)\ );$
$return\ rc$
od
$);$
$return\ rc$
od

definition $smack\text{-}socket\text{-}post\text{-}create :: State' \Rightarrow smacksocket \Rightarrow Sk\text{-}Family$
$\Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow (State', int)$
$nondet\text{-}monad$
where $smack\text{-}socket\text{-}post\text{-}create\ s\ sock\ family\ type'\ protocols\ kern \equiv do$
$ssp \leftarrow return(sk\ sock);$
$rc \leftarrow (if\ sk\ sock =\ None\ then\ return\ 0$
$else$
$if\ family \neq PF\text{-}INET\ then$
$return\ 0$
$else$
$smack\text{-}netlabel\ s\ (the(sk\ sock))\ SMACK\text{-}CIPSO\text{-}SOCKET$
$);$
$return\ rc$
od

definition *smack-socket-socketpair* :: *State'* ⇒ *smacksocket* ⇒ *smacksocket* ⇒ (*State'*, *int*) *nondet-monad*
  where *smack-socket-socketpair s socka sockb* ≡ *do*
    asp ← *return*(*the*(*sk-security s*(*the*(*sk socka*))));
    bsp ← *return*(*the*(*sk-security s* (*the*(*sk sockb*))));
    ask ← *return*(*the*(*sk socka*));
    bsk ← *return*(*the*(*sk sockb*));
      modify(λ*s* .*s*(|*sk-security* := (*sk-security s*)(*ask* := *Some*(*asp*(|*smk-packet* := *Some*(*smk-out bsp*)|)))|));
      modify(λ*s* .*s*(|*sk-security* := (*sk-security s*)(*bsk* := *Some*(*bsp*(|*smk-packet* := *Some*(*smk-out asp*)|)))|));
    rc ← *return*(*0*);
    return rc
    od

definition *smk-ipv6-port-label* :: *State'* ⇒ *smacksocket* ⇒ *sockaddr* ⇒ (*State'*, *unit*) *nondet-monad*
  where *smk-ipv6-port-label s sock address* ≡ *return* ()

definition *smack-socket-bind* :: *State'* ⇒ *socket* ⇒ *sockaddr* ⇒*int*⇒ (*State'*, *int*) *nondet-monad*
  where *smack-socket-bind s sock address addrlen* ≡ *do*
    socka ← *return*(*sk sock*);
     if socka ≠ *None* ∧ (*sk-family* (*the*(*sk sock*))) = *PF-INET6* *then*
    do
      smk-ipv6-port-label s sock address;
      return 0
    od
    else
       return 0
    od

definition *ipv4host-label-find* :: *nat* ⇒*in-addr*⇒ (*State'*,*smack-known option*) *nondet-monad*
  where *ipv4host-label-find a' siap* ≡
  do (*a'*, *result*) ← *whileLoop*
    (λ(*a'*, *result*) *siap*. *a'* < *length*(*smk-net4addr-list*))
    (λ(*a'* ,*result*) . ((*if* (*int* (*s-addr* (*net4-smk-host* (*smk-net4addr-list* ! *a'*)))) =
        (*int*(*s-addr siap*) *AND* (*int*(*s-addr* (*net4-smk-mask* (*smk-net4addr-list* ! *a'*)))))
      then return (*a'* + *1*, *Some* (*net4-smk-label* (*smk-net4addr-list* ! *a'*)))
      else return (*a'* + *1*, *None*))))
        (*a'*, *None*);
  return result

od

definition *smack-ipv4host-label* :: *State'* ⇒ *sockaddr-in* ⇒ (*State'*, *smack-known option*) *nondet-monad*
  where *smack-ipv4host-label s sip* ≡ *do*
    siap ← *return*(*sin-addr sip*);
    rc ← *ipv4host-label-find 0 siap*;
    return rc
    od

definition *smack-netlabel-send* :: *State'* ⇒ *sock* ⇒ *sockaddr-in* ⇒ (*State'*, *int*) *nondet-monad*
  where *smack-netlabel-send s sock sap* ≡ *do*
    skp ← *return* (*SOME x* :: *smack-known* .*True*);
   hkp ← *return* (*SOME x* :: *smack-known* .*True*);
    ssp ← *return*(*the*(*sk-security s sock*));
    ad ← *return*(*SOME x* :: *smk-audit-info* .*True*);
    hkp ← *smack-ipv4host-label s sap*;
    rc ← (*if hkp* ≠ *None then*
      do
        sk-lbl ← *return SMACK-UNLABELED-SOCKET*;
        skp ← *return* (*smk-out ssp*);
          rc ← *smk-access s skp* (*the hkp*) *MAY-WRITE* (*Some ad*);
         rc ← *return*(*smk-bu-note s ''IPv4 host check'' skp* (*the hkp*) *MAY-WRITE rc*);
        if rc ≠ *0 then return rc*
        else smack-netlabel s sock sk-lbl
      od
      else do
          sk-lbl ← *return SMACK-CIPSO-SOCKET*;
          smack-netlabel s sock sk-lbl
        od
     );
    return rc

    od

definition *smk-ipv6-localhost* :: *sockaddr-in6* ⇒ *bool*
  where *smk-ipv6-localhost sip* ≡ *True*

definition *smack-ipv6host-label* :: *State'* ⇒ *sockaddr-in6* ⇒ (*State'*, *smack-known option*) *nondet-monad*
  where *smack-ipv6host-label s sip* ≡ *do*
    rc ← *return* (*SOME x* :: *smack-known option* .*True*);
    rc ← (*if smk-ipv6-localhost sip then return* (*None*)
      else return rc);
    return rc

od

definition *smk-ipv6-check* :: *State′* ⇒ *smack-known* ⇒ *smack-known*
⇒ *sockaddr-in6* ⇒ *int* ⇒ (*State′*, *int*) *nondet-monad*
where *smk-ipv6-check s subj obj addr act* ≡ *do*

```
ad ← return( SOME x :: smk-audit-info .True);
rc ← smk-access s subj  obj  MAY-WRITE (Some ad);
rc ← return(smk-bu-note s ''IPv6 check'' subj obj MAY-WRITE rc);
return rc
od
```

definition *smk-ipv6-port-check* :: *State′* ⇒ *sock* ⇒*sockaddr-in6* ⇒ *int* ⇒ (*State′*, *int*) *nondet-monad*
where *smk-ipv6-port-check s sock addr act* ≡ *do*

```
ad ← return( SOME x :: smk-audit-info .True);
ssp ← return(the(sk-security s sock));
skp ← (if act = SMK-RECEIVING then smack-ipv6host-label s addr
         else return (Some(smk-out ssp)));
obj ← (if act = SMK-RECEIVING then return (Some(smk-in ssp))
         else smack-ipv6host-label s addr);

rc ← (if skp = None ∧ obj = None
       then smk-ipv6-check s (the skp) (the obj) addr act
       else
         do
           skp ← (if skp = None then
                     return (smack-net-ambient shared)
                   else
                     return (the skp)
                  );
           obj ← (if obj = None then
                     return (smack-net-ambient shared)
                   else
                     return (the obj));
           rc ←(if(¬(smk-ipv6-localhost addr)) then
                   smk-ipv6-check s ( skp) ( obj) addr act
                 else if act =  SMK-RECEIVING
                      then return 0
                     else
                      smk-ipv6-check s skp obj addr act
            );
           return rc
         od
       );
return rc
```

od

definition *sockaddr-to-sockaddr-in* :: *sockaddr* => *sockaddr-in*
  where *sockaddr-to-sockaddr-in sap* ≡ (*SOME x* :: *sockaddr-in* .*True*)

definition *smack-socket-connect* :: *State′* ⇒ *smacksocket* ⇒ *sockaddr*⇒*int* ⇒ (*State′*, *int*) *nondet-monad*
  where *smack-socket-connect s sock sap addrlen* ≡ *do*
    rc ← *return*(*0*);
    sk ← *return* (*sk sock*);
    ssp ← *return*(*the*(*sk-security s*(*the*(*sk*))));
    sap ← *return*(*SOME x* :: *sockaddr-in* .*True*);
    sip ← *return*(*SOME x* :: *sockaddr-in6* .*True*);
    rc ← (*if sk* = *None then return 0 else do*
      sk-family ← *return*(*sk-family* (*the*(*sk*)));
      case sk-family of PF-INET ⇒
        do
          ret ← *smack-netlabel-send s* (*the sk*) *sap*;
          return ret
        od | *PF-INET6* ⇒
        do
          rsp ← *smack-ipv6host-label s sip*;
          ret ← *smk-ipv6-check s* (*smk-out ssp*) (*the rsp*)
sip SMK-CONNECTING;
          return ret
        od | - ⇒ *return rc*
    od);
    return rc
    od

definition *getSockaddr-in* :: *Msghdr-name option* ⇒ *sockaddr-in option*
  where *getSockaddr-in name* ≡ *let e* = *SOME e*. *Sockaddr-in e* = *the name in Some e*

definition *getSockaddr-in6 name* ≡ *let e* = *SOME e*. *Sockaddr-in6 e* = *the name in Some e*
term *getSockaddr-in* (*msg-name msg*)

definition *smack-socket-sendmsg* :: *State′* ⇒ *smacksocket* ⇒ *msghdr*⇒*int* ⇒ (*State′*, *int*) *nondet-monad*
  where *smack-socket-sendmsg s sock msg size′* ≡ *do*
    rc ← *return*(*0*);
    sip ← *return*(*getSockaddr-in* (*msg-name msg*));
    sap ← *return* (*getSockaddr-in6* (*msg-name msg*));
    sk ← *return* (*the*(*sk sock*));
    ssp ← *return*(*the*(*sk-security s*(*sk*)));

sk-family ← *return*(*sk-family sk*);
rc ← (*if sip* = *None then return 0 else*

case sk-family of PF-INET ⇒
do
ret ← *smack-netlabel-send s sk* (*the sip*);
return ret
od | *PF-INET6* ⇒
do
rc ←(*if SMACK-IPV6-SECMARK-LABELING conf then*
do
rsp ← *smack-ipv6host-label s* (*the sap*);
if rsp ≠ *None then*
do
ret ← *smk-ipv6-check s* (*smk-out ssp*) (*the rsp*) (*the sap*) *SMK-CONNECTING*;
return ret od else return rc
od
else return rc);
rc ← (*if SMACK-IPV6-PORT-LABELING conf then*
do
ret ← *smk-ipv6-port-check s sk* (*the sap*) *SMK-SENDING*;
return ret
od
else
return rc
);
return rc
od | - ⇒ *return rc*

);
return rc

od

definition *netlbl-skbuff-getattr* :: *sk-buff* ⇒ *Sk-Family* ⇒ *netlbl-lsm-secattr* ⇒ *int*
where *netlbl-skbuff-getattr skb family secattr* ≡ (−*ENOSYS*)

definition *smack-socket-sock-rcv-skb* :: *State′* ⇒ *sock* ⇒ *sk-buff*⇒ (*State′*, *int*) *nondet-monad*
where *smack-socket-sock-rcv-skb s sock skb* ≡ *do*
rc ← *return*(*0*);
ssp ← *return*(*the*(*sk-security s sock*));
sk-family ← *return*(*sk-family sock*);

ad ← *return*(*SOME x* :: *smk-audit-info* .*True*);
sadd ← *return*(*SOME x* :: *sockaddr-in6* .*True*);
secattr ← *return*(*SOME x* :: *netlbl-lsm-secattr* .*True*);
family ←(*if sk-family* = *PF-INET6* ∧ *protocol skb* = *ETH-P-IP*

then return(*PF-INET*)
else return sk-family
);
rc ← (
case family of PF-INET ⇒
if CONFIG-SECURITY-SMACK-NETFILTER conf then
if secmark skb ≠ *0 then*
do
skp← *smack-from-secid* (*secmark skb*);
rc ← *smk-access s* (*the skp*) (*smk-in ssp*) *MAY-WRITE* (*Some ad*);
rc ← *return*(*smk-bu-note s ''IPv4 delivery''* (*the skp*) (*smk-in ssp*) *MAY-WRITE rc*);
return rc
od
else
return (*netlbl-skbuff-getattr skb family secattr*)

else
return (*netlbl-skbuff-getattr skb family secattr*)

| *PF-INET6* ⇒
do
skp← (*if SMACK-IPV6-SECMARK-LABELING conf then*
if (*secmark skb*) ≠ *0 then smack-from-secid* (*secmark skb*)
else
smack-ipv6host-label s sadd
else return(*None*));
skp ← (*if skp* = *None then return* (*smack-net-ambient shared*)
else return (*the skp*));
rc ← (*if SMACK-IPV6-SECMARK-LABELING conf then*
do
rc ← *smk-access s* (*skp*) (*smk-in ssp*) *MAY-WRITE* (*Some ad*);
rc ← *return*(*smk-bu-note s ''IPv6 delivery''* (*skp*) (*smk-in ssp*) *MAY-WRITE rc*);
return rc
od
else if SMACK-IPV6-PORT-LABELING conf

then smk-ipv6-port-check s sock sadd
SMK-RECEIVING
else return rc
);
return rc
od | - ⇒ *return rc*

);
return rc
od

definition *smack-copy-to-user* :: *string* => *string* => *nat* => *int*
where*smack-copy-to-user from to n* ≡ *let to* = *take n from in if*(*length to*)= *0* *then 1 else 0*

definition *smack-put-user* :: *int* => *int* => *int*
where*smack-put-user x ptr* ≡ *let x* = *ptr in 0*

definition *smack-socket-getpeersec-stream* :: *State′* ⇒ *socket* ⇒ *string*⇒*int* ⇒*nat* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-socket-getpeersec-stream s sock optval optlen len′*≡ *do*
ssp ← *return*(*the*(*sk-security s* (*the*(*sk sock*))));
rcp ← *return*(*′′′′*);
slen ← *return*(*1*);
rc ← *return* (*0*);
sk ← *return* (*sk sock*);
sk-family ← *return*(*sk-family* (*the sk*));
rcp ← (*if* (*smk-packet ssp*) ≠ *None*
then return (*smk-known* (*the*(*smk-packet ssp*)))
else return rcp);
slen ← (*if* (*smk-packet ssp*) ≠ *None*
then return (*length*((*smk-known* (*the*(*smk-packet* *ssp*))))+*1*)
else return slen);
rc ← (*if slen* > *len′ then return* (−*ERANGE*)
else if (*smack-copy-to-user optval rcp slen*) ≠ *0*
then return (−*EFAULT*)
else if (*smack-put-user slen optlen* ≠ *0*)
then return (−*EFAULT*)
else return rc
);
return rc
od

definition *smack-from-secattr* :: *State′* ⇒ *netlbl-lsm-secattr* ⇒ *socket-smack* ⇒

$(State', smack\text{-}known\ )\ nondet\text{-}monad$
where *smack-from-secattr s sap ssp ≡ do*
rc ← return(SOME x :: smack-known .True);
return rc
od

definition *smack-socket-getpeersec-stream-t* ::
State′ ⇒ socket ⇒ Sk-Family ⇒netlbl-lsm-secattr ⇒sk-buff ⇒ (State′, int) nondet-monad
where *smack-socket-getpeersec-stream-t s sock family secattr skb≡ do*
sid ← (if sk sock ≠ None then
do
ssp ← return(the(sk-security s (the(sk sock))));

rc ← return (netlbl-skbuff-getattr skb family secattr);
if rc = 0 then do skp← smack-from-secattr s secattr ssp;
return (smk-secid skp)od else return 0
od
else return (0));
return sid
od

definition *smack-socket-getpeersec-dgram* :: *State′ ⇒ socket ⇒ sk-buff option⇒u32 ⇒ (State′, int) nondet-monad*
where *smack-socket-getpeersec-dgram s sock skb secid′≡ do*
ssp ← return(the(sk-security s (the(sk sock))));
family ← return(PF-UNSPEC);
skb ← return (the skb);
family ← (if (protocol skb) = ETH-P-IP then return PF-INET

else if(CONFIG-IPV6 conf ∧ (protocol skb) = ETH-P-IPV6)
then return PF-INET6
else if (family = PF-UNSPEC) then
return (sk-family (the(sk sock)))
else return family
);
secattr ← return(SOME x :: netlbl-lsm-secattr .True);
sid ← (case family of PF-UNIX ⇒ return(smk-secid (smk-out ssp)) |
PF-INET ⇒
(if CONFIG-SECURITY-SMACK-NETFILTER conf then
do
sid ← return(secmark skb);
if (sid ≠ 0) then return sid
else smack-socket-getpeersec-stream-t s sock family

secattr skb
od
else
smack-socket-getpeersec-stream-t s sock family secattr skb)|
PF-INET6 ⇒ *if SMACK-IPV6-SECMARK-LABELING conf then return* (*secmark skb*)
else return 0
);
secid ← *return sid*;
rc ← *if sid* = *0 then return*(−*EINVAL*)
else return (*0*);
return rc
od

definition *smack-sk-alloc-security* :: *State′* ⇒ *sock* ⇒ *int*⇒ *gfp-t* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-sk-alloc-security s sock family flgs*≡ *do*
skp ← *return*(*smk-of-current s*);
ssp ← *return*(*sk-security s* (*sock*));
rc ← *return* (*0*);
f ← *return* (*flags* (*get-cur-task s*));
rc← (*if ssp* = *None then return* (−*ENOMEM*)
else if (*f* = *PF-KTHREAD*) *then*
do

modify(λ*s* .*s*⦇*sk-security* := (*sk-security s*)(*sock* :=
Some(⦇*smk-out* = *smack-known-web*,
smk-in = *smack-known-web*,
smk-packet = *None* ⦈))⦈);
return rc
od
else
do

modify(λ*s* .*s*⦇*sk-security* := (*sk-security s*)(*sock* :=
Some(⦇*smk-out* = *smack-known-web*,
smk-in = *smack-known-web*,
smk-packet = *None* ⦈))⦈);
return rc
od
);

return rc
od

definition *smack-sk-free-security* :: *State′* ⇒ *sock* ⇒ (*State′*, *unit*) *nondet-monad*
where *smack-sk-free-security s sock* ≡ *do*
modify(λ*s* .*s*⦇*sk-security* := (*sk-security s*)(*sock* := *None*)⦈);

return()
od

definition *smack-sock-graft* :: *State′* ⇒ *sock* ⇒ *socket* ⇒ (*State′*, *unit*) *nondet-monad*
where *smack-sock-graft s sock parent′*≡ *do*
ssp ← *return*(*the*(*sk-security s* (*sock*)));
skp ← *return*(*smk-of-current s*);
rc ← (*if sk-family sock* ≠ *PF-INET* ∧ *sk-family sock* ≠ *PF-INET6*
then return()
else do
modify(λ*s* .*s*(|*sk-security* := (*sk-security s*)(*sock* := *Some*(*ssp*(|*smk-in* := *skp*,*smk-out* := *skp* |))) |));
return()
od
);

return rc
od

definition *netlbl-req-setattr* :: *request-sock* => *netlbl-lsm-secattr* ⇒ *int*
where *netlbl-req-setattr req secattr* ≡ (−*ENOSYS*)

definition *smack-inet-conn-request* :: *State′* ⇒ *sock* ⇒ *sk-buff* ⇒ *request-sock* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-inet-conn-request s sock skb req*≡ *do*
family ← *return*(*sk-family sock*);
ssp ← *return*(*the*(*sk-security s* (*sock*)));
ad ← *return*(*SOME x* :: *smk-audit-info* .*True*);
secattr ← *return*(*SOME x* :: *netlbl-lsm-secattr* .*True*);
family ← (*if CONFIG-IPV6 conf* ∧ *family* = *PF-INET6* ∧ *protocol skb* = *ETH-P-IP*
then return(*PF-INET*) *else return family*);
rc ← (*if* ¬(*CONFIG-IPV6 conf* ∧ *family* = *PF-INET6* ∧ *protocol skb* = *ETH-P-IP*)
then return(*0*)
else
if CONFIG-SECURITY-SMACK-NETFILTER conf then do
skp← (*if secmark skb* ≠ *0 then*
smack-from-secid (*secmark skb*)
else return (*None*));
rc ← *smk-access s* (*the skp*) (*smk-in ssp*) *MAY-WRITE* (*Some ad*);
rc ← *return*(*smk-bu-note s ′′IPv4 connect′′* (*the skp*) (*smk-in ssp*) *MAY-WRITE rc*);

rc ← (*if rc* ≠ *0 then return rc else*
do
addr ← *return*(*SOME x*:: *sockaddr-in*.
True);

hskp ← *smack-ipv4host-label s addr* ;
if hskp = *None*
then
return(*netlbl-req-setattr req*
(*smk-netlabel* (*the skp*)))
else
return rc
od
);
return rc
od
else
return 0
);

return rc
od

definition *smack-inet-csk-clone* :: *State′* ⇒ *sock* ⇒ *request-sock* ⇒ (*State′*, *unit*)
nondet-monad
where *smack-inet-csk-clone s sock req* ≡ *do*
ssp ← *return*(*the*(*sk-security s* (*sock*)));
skp ← *return*(*smk-of-current s*);
(*if peer-secid req* ≠ *0*
then do skp ← *smack-from-secid* (*peer-secid req*);
modify(λ*s* .*s*⦇*sk-security* := (*sk-security s*)(*sock*
:=
Some(*ssp*⦇*smk-packet* := *skp* ⦈)⦈) *od*
else
modify(λ*s* .*s*⦇*sk-security* := (*sk-security s*)(*sock* :=
Some(*ssp*⦇*smk-packet* := *None* ⦈)⦈)

);
return ()
od

29.53 audit hook

definition *smack-audit-rule-init* ::
State′ ⇒ *u32* ⇒ *enum-audit* ⇒ *string* ⇒ *string*⇒ (*State′*, *int*) *nondet-monad*
where *smack-audit-rule-init s field op rulestr vrule* ≡ *do*
skp ← *smk-import-entry s rulestr 0*;

$rc \leftarrow$ (*if field* $\neq$ *AUDIT-SUBJ-USER* $\wedge$ *field* $\neq$ *AUDIT-OBJ-USER*

then return ($-$*EINVAL*)
else if op $\neq$ *Audit-equal* $\wedge$ *op* $\neq$ *Audit-not-equal then return* ($-$*EINVAL*)
else if skp = None then return ($-$*ENOMEM*)
else do rule $\leftarrow$ *return*(*smk-known* (*the*(*skp*)));
return 0
od
);
return rc
od

definition *smack-audit-rule-known* :: *State'* $\Rightarrow$ *audit-krule* $\Rightarrow$ (*State'*, *int*) *nondet-monad*
where *smack-audit-rule-known s krule* $\equiv$
do
$a' \leftarrow$ *return*(*0*);
(a', *result*) $\leftarrow$ *whileLoop*
(λ(a', *result*) *secid'*. a' < (*field-count krule*))
(λ(a' ,*result*) . ((*if atype* ((*afields krule*) ! a') = *AUDIT-SUBJ-USER* $\vee$
atype ((*afields krule*) ! a') = *AUDIT-OBJ-USER*
then return (a' + *1*, *1*)
else return (a' + *1*, *0*))))
(a', *0*);
return result
od

definition *smack-audit-rule-match* ::
State' $\Rightarrow$ *u32* $\Rightarrow$ *u32* $\Rightarrow$ *enum-audit* $\Rightarrow$ *string* $\Rightarrow$ *audit-context* $\Rightarrow$ (*State'*, *int*) *nondet-monad*
where *smack-audit-rule-match s secid' field op vrule actx* $\equiv$ *do*
rule $\leftarrow$ *return*(*vrule*);
rc $\leftarrow$ (*if unlikely* (*length*(*rule*)) *then return*($-$*ENOENT*) *else*
if field $\neq$ *AUDIT-SUBJ-USER* $\wedge$ *field* $\neq$ *AUDIT-OBJ-USER* *then return 0*
else do
skp $\leftarrow$ *smack-from-secid secid'*;
if op = Audit-equal then if rule = smk-known (*the skp*)
then return 1
else return 0
else if op = Audit-not-equal then if rule $\neq$ *smk-known* (*the skp*)
then return 1 else return 0
else return 0
od
);

return rc
od

29.54 other

definition *smack-getprocattr* :: *State'* ⇒ *Task* ⇒ *string* ⇒ *string* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-getprocattr s p name value* ≡ *do*
skp ← *return* (*smk-of-task-struct s p*);
cp ← *return*(*kstrdup* (*smk-known skp*));
rc ← (*if length*(*the cp*) = *0 then return* (*uminus ENOMEM*)
else return (*length* (*the cp*)));
return rc
od

definition *smack-d-instantiate* :: *State'* ⇒ *dentry* ⇒ *inode option* ⇒ (*State'*, *unit*) *nondet-monad*
where *smack-d-instantiate s opt-dentry inode* ≡ *do*
skp ← *return* (*smk-of-current s*);

rc ← (*if inode* = *None then return* ()
else do
isp ← *return*(*the*(*i-security s* (*the inode*)));
if (*smk-iflags isp AND SMK-INODE-INSTANT*)
≠ *0 then return*()
else do
sbp ← *return*(*i-sb* (*the inode*));
sbsp ← *return*(*the*(*sb-security s sbp*));
return()
od
od

);
return rc
od

definition *smack-setprocattr-known* :: *State'* ⇒ *smack-known list* ⇒*smack-known*⇒ (*State'*, *int*) *nondet-monad*
where *smack-setprocattr-known s relabellist skp* ≡
do
a' ← *return*(*0*);
(*a'*, *result*) ← *whileLoop*
(λ(*a'*, *result*) *secid'*. *a'* < (*length relabellist*))
(λ(*a'* ,*result*) . ((*if* (*relabellist* ! *a'*) = *skp*
then return (*a'* + *1*, *0*)
else return (*a'* + *1*, (−*EPERM*)))))
(*a'*, (−*EPERM*));
return result
od

definition *smack-setprocattr* :: *State′* ⇒ *string* ⇒ *string* ⇒ *int* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-setprocattr s name value size′*≡ *do*
tsp ← *return*(*current-security s*);
rc ← (*if* ¬(*smack-privileged s CAP-MAC-ADMIN*) ∧ *length*(*smk-relabel tsp*) = *0*
then return (*uminus EPERM*)
else
if length(*value*) =*0* ∨ *size′* = *0* ∨ *size′* ≥ *SMK-LONGLABEL*

then return (−*EINVAL*)
else if name ≠ ′′*current*′′
then return (−*EINVAL*) *else*
do
skp ← *smk-import-entry s value size′*;
if skp = *None then return* (−*ENOMEM*)
else if (*the skp*) = *smack-known-web* ∨ (*the skp*) = *smack-known-star*
then return (−*EINVAL*)
else if ¬(*smack-privileged s CAP-MAC-ADMIN*) *then*
do
rc ← *smack-setprocattr-known s* (*smk-relabel tsp*) (*the skp*);
return rc
od
else
do
new ← *return*(*snd*(*prepare-creds s*));
if new = *None then return* (−*ENOMEM*) *else*
return size′
od
od
);
return rc
od

definition *smack-ismaclabel* :: *State′* ⇒ *xattr* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-ismaclabel s name* ≡ *do*

rc ← (*if name* = *XATTR-SMACK-SUFFIX then return* (*1*)
else return (*0*));
return rc
od

definition *smack-secid-to-secctx* :: *State′* ⇒ *u32* ⇒*string* ⇒*u32* ⇒ (*State′*, *int*)

nondet-monad
  where *smack-secid-to-secctx s secid′ secdata seclen* ≡ *do*
      skp ← *smack-from-secid secid′*;
        secdata ← (*if length*(*secdata*) ≠ *0 then return* (*smk-known* (*the skp*))
          else return secdata);
      seclen ← *return* (*length*(*smk-known* (*the skp*)));
      return 0
      od

definition *smack-secctx-to-secid* :: *State′* ⇒*string* ⇒*u32* ⇒*u32*⇒ (*State′*, *int*) *nondet-monad*
  where *smack-secctx-to-secid s secdata seclen secid′* ≡ *do*
      skp ← *smk-find-entry secdata*;
      secid′ ← (*if skp* = *None then return* (*smk-secid* (*the skp*))
          else return (*0*));
      return 0
      od

definition *smack-inode-notifysecctx* :: *State′* ⇒*inode* ⇒*string* ⇒*u32*⇒ (*State′*, *int*) *nondet-monad*
  where *smack-inode-notifysecctx s inode ctx ctxlen* ≡
    smack-inode-setsecurity s inode XATTR-SMACK-SUFFIX (*String ctx*) *ctxlen 0*

definition *vfs-setxattr* :: *State′* ⇒ *dentry* ⇒*string* ⇒*string* ⇒ *int* ⇒ *int*⇒ (*State′*, *int*) *nondet-monad*
  where *vfs-setxattr s dentry name value size′ flgs* ≡ *return 0*

definition *is-bad-inode* :: *inode* ⇒ *bool*
  where *is-bad-inode inode* ≡ *True*

definition *vfs-setxattr-noperm′* :: *State′* ⇒ *dentry* ⇒*string* ⇒*string* ⇒ *int* ⇒ *int*⇒ (*State′*, *int*) *nondet-monad*
  where *vfs-setxattr-noperm′ s dentry name value size′ flgs* ≡ *do*
    inode ← *return*(*get-inode s* (*d-inode dentry*));
    error ← *return* (−*EAGAIN*);
    inode ← (*if name* ≠ ″*security.*″ *then do*
        f ← *return* ((*int*(*i-flags* (*the inode*))) *AND* (*NOT S-NOSEC*));
          indoe ← *return*((*the inode*)⦇ *i-flags* := (*nat f*)⦈);
          return (*inode*)
          od
          else return(*inode*));
    error ← (*if* (*int*(*i-opflags* (*the inode*)) *AND IOP-XATTR*) ≠ *0*

then

vfs-setxattr s dentry name value size′ flgs

else

if is-bad-inode (*the inode*) *then return*(−*EIO*)
else if (*error* = (−*EAGAIN*)) *then*
return (−*EOPNOTSUPP*)
else return error

);
return error
od

definition *smack-inode-setsecctx* :: *State′* ⇒*dentry* ⇒*string* ⇒*u32*⇒ (*State′*, *int*) *nondet-monad*
where *smack-inode-setsecctx s dentry ctx ctxlen* ≡
vfs-setxattr-noperm′ s dentry ′′security.SMACK64′′ ctx ctxlen 0

definition *smack-inode-getsecctx* :: *State′* ⇒*inode* ⇒*string* ⇒*u32*⇒ (*State′*, *int*) *nondet-monad*
where *smack-inode-getsecctx s inode ctx ctxlen* ≡ *do*
skp ← *return*(*smk-of-inode s inode*);
ctx ← *return*(*smk-known skp*);
ctxlen ← *return*(*length*(*smk-known skp*));
return 0
od

definition *smack-inode-copy-up* :: *State′* ⇒ *dentry* ⇒*Cred option* ⇒ (*State′*, *int*) *nondet-monad*
where *smack-inode-copy-up s dentry new*≡ *do*
new-creds ← *return*(*new*);
rc ← (*if new-creds* = *None then*
do
new-creds ← *return*(*snd*(*prepare-creds s*));
if new-creds = *None then return* (−*ENOMEM*) *else*
do
tsp ← *return*(*t-security s* (*the new-creds*));
isp ← *return* (*i-security s* (*the*(*get-inode s* (*d-inode* (*the*(*get-dentry s* (*d-parent dentry*))))))));
skp ← *return*(*smk-inode* (*the isp*));
modify(*λs .s*⦇*t-security* := (*t-security s*)((*the new-creds*) := *Some* (*the tsp* ⦇ *smk-task* := *skp* ⦈))⦈);
new ← *return*(*new-creds*);
return 0
od
od

else
return 0);

return rc
od

definition *smack-inode-copy-up-xattr* :: *State'* ⇒*xattr* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-inode-copy-up-xattr s name* ≡ *if name* = *XATTR-NAME-SMACK then return 1*
else return (−*EOPNOTSUPP*)

definition *smack-dentry-create-files-as* :: *State'* ⇒ *dentry* ⇒*mode* ⇒*string* ⇒*Cred* ⇒*Cred* ⇒ (*State'*, *int*) *nondet-monad*
where *smack-dentry-create-files-as s dentry mode' name old new*≡ *do*
otsp ← *return*(*the*(*t-security s old*));
ntsp ← *return*(*the*(*t-security s new*));
modify(λ*s* .*s*⦇*t-security* := (*t-security s*)(*new* := *Some* (*ntsp* ⦇ *smk-task* := *smk-task otsp* ⦈))⦈);
isp ← *return* (*the*(*i-security s* (*the*(*get-inode s* (*d-inode* (*the*(*get-dentry s* (*d-parent dentry*))))))));
if (*smk-iflags isp AND SMK-INODE-TRANSMUTE*) ≠ *0 then*
do
may ← *smk-access-entry s* (*smk-known* (*smk-task otsp*)) (*smk-known* (*smk-inode isp*)) (*smk-rules* (*smk-task otsp*));
(*if may* > *0* ∧ ((*may AND MAY-TRANSMUTE*) ≠ *0*) *then*
modify(λ*s* .*s*⦇*t-security* := (*t-security s*)(*new* := *Some* (*ntsp* ⦇ *smk-task* := *smk-task otsp* ⦈))⦈)
else
modify(λ*s* .*s*⦇*t-security* := (*t-security s*)(*new* := *Some* (*ntsp* ⦇ *smk-task* := *smk-inode isp* ⦈))⦈));
return 0
od
else return 0
od

definition *smack-init* :: *State'*⇒(*State'*, *int*) *nondet-monad*
where *smack-init s* ≡ *do*
cred' ← *return* (*SOME x* :: *Cred* .*True*);
tsp ← *return* (*SOME x* :: *task-smack* .*True*);
cred' ← *return* (*current-cred* (*current-task s*));

return(*0*)
od

end

30 Smack proof

theory *SmackLemma*
imports
SmackHooks
begin

30.1 Correctness for Smack TDS specification

lemma *smk-access-entry-not-chg-state*:
$\bigwedge s'$.
$\{\!|\lambda s.\ s = s'|\!\}$
smk-access-entry s' *subj obj r*
$\{\!|\lambda r\ s.\ s = s'\ |\!\}$
apply(*unfold smk-access-entry-def*)
apply *wpsimp*
done

30.2 correctness lemmas of smack$_{p}trace_{a}ccess_{c}heck$

lemma *smack-ptrace-access-check-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-ptrace-access-check s t m* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0|\!\}$
apply(*unfold smack-ptrace-access-check-def smk-ptrace-rule-check-def*)
apply *wpsimp*
done

lemma *smack-ptrace-access-check-correctness1*:
$\exists\, t\ m.$ $\{\!|\lambda s.$ *smk-known* (*smk-of-task*(*the*((*t-security s*)(*task-cred* (*current-task s*)))))
= *smk-known* (*smk-of-task-struct s t*)
$\wedge$ (((*int m*) *AND PTRACE-MODE-ATTACH*) $\neq$ *0*) $\wedge$ (
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-EXACT*) $\vee$
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-DRACONIAN*))
= *True*
$|\!\}$
smack-ptrace-access-check s t m $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
apply(*unfold smack-ptrace-access-check-def smk-ptrace-rule-check-def*)
apply *wpsimp*
by (*metis int-and-0 semiring-1-class.of-nat-0*)

lemma *smack-ptrace-access-check-correctness2*:
$\exists\, t\ m.$ $\{\!|\lambda s.$ *smk-known* (*smk-of-task*(*the*((*t-security s*)(*task-cred* (*current-task s*)))))
$\neq$ *smk-known* (*smk-of-task-struct s t*)
$\wedge$ (((*int m*) *AND PTRACE-MODE-ATTACH*) $\neq$ *0*) $\wedge$ (
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-EXACT*) $\vee$
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-DRACONIAN*))
= *True*
$\wedge$ (*smack-ptrace-rule shared*) = *SMACK-PTRACE-DRACONIAN*
$|\!\}$
smack-ptrace-access-check s t m $\{\!|\lambda r\ s.\ r = -EACCES\ |\!\}$
apply(*unfold smack-ptrace-access-check-def smk-ptrace-rule-check-def*)

apply *wpsimp*
by (*metis int-and-0 semiring-1-class.of-nat-0*)

lemma *smack-ptrace-access-check-correctness3*:
$\exists t\, m.$ $\{\!|\lambda s.$ *smk-known* (*smk-of-task*(*the*((*t-security s*)(*task-cred* (*current-task s*)))))
$\neq$ *smk-known* (*smk-of-task-struct s t*)
$\wedge$ (((*int m*) *AND PTRACE-MODE-ATTACH*) $\neq$ *0*) $\wedge$ (
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-EXACT*) $\vee$
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-DRACONIAN*))
= *True*
$\wedge$ (*smack-ptrace-rule shared*) $\neq$ *SMACK-PTRACE-DRACONIAN*
$\wedge$ *smack-privileged-cred CAP-SYS-PTRACE tracercred*
$|\!\}$
smack-ptrace-access-check s t m $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
apply(*unfold smack-ptrace-access-check-def smk-ptrace-rule-check-def*)
apply *wpsimp*
by (*metis int-and-0 semiring-1-class.of-nat-0*)

lemma *smack-ptrace-access-check-correctness4*:
$\exists t\, m.$ $\{\!|\lambda s.$ *smk-known* (*smk-of-task*(*the*((*t-security s*)(*task-cred* (*current-task s*)))))
$\neq$ *smk-known* (*smk-of-task-struct s t*)
$\wedge$ (((*int m*) *AND PTRACE-MODE-ATTACH*) $\neq$ *0*) $\wedge$ (
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-EXACT*) $\vee$
((*smack-ptrace-rule shared*) = *SMACK-PTRACE-DRACONIAN*))
= *True*
$\wedge$ (*smack-ptrace-rule shared*) $\neq$ *SMACK-PTRACE-DRACONIAN*
$\wedge$ *smack-privileged-cred CAP-SYS-PTRACE tracercred* = *Fasle*
$|\!\}$
smack-ptrace-access-check s t m $\{\!|\lambda r\ s.\ r = -EACCES\ |\!\}$
apply(*unfold smack-ptrace-access-check-def smk-ptrace-rule-check-def*)
apply *wpsimp*
by (*metis int-and-comm int-and-extra-simps*(*1*) *semiring-1-class.of-nat-0*)

30.3 correctness lemmas of smack$_{p}trace_{t}raceme$

lemma *smack-ptrace-traceme-correctness*:
$\{\!|\lambda s.$ *True*$|\!\}$ *smack-ptrace-traceme s ptp* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0\ |\!\}$
apply(*unfold smack-ptrace-traceme-def smk-ptrace-rule-check-def*)
apply *wpsimp*
done

30.4 correctness lemmas of smack$_{s}yslog$

lemma *smack-syslog-correctness*:
$\{\!|\lambda s.$ *True*$|\!\}$ *smack-syslog s t* $\{\!|\lambda r\ s.\ r = 0 \vee r =$ *uminus EACCES* $|\!\}$
apply(*unfold smack-syslog-def*)
apply *wpsimp*
done

lemma *smack-syslog-correctness1*:
$\exists s.$ $\{\!|\lambda s.$ *smack-privileged s CAP-MAC-OVERRIDE* $=$ *True*$|\!\}$ *smack-syslog s t* $\{\!|\lambda r$ $s.$ $r = 0$ $|\!\}$
apply(*unfold smack-syslog-def*)
apply *wpsimp*
by (*metis* (*mono-tags*, *lifting*) *hoare-return-simp*)

lemma *smack-syslog-correctness2*:
$\exists s.$ $\{\!|\lambda s.$ *smack-privileged s CAP-MAC-OVERRIDE* $=$ *False*
$\wedge$ *smk-of-current s* $\neq$ *smack-syslog-label shared* $|\!\}$ *smack-syslog s t* $\{\!|\lambda r$ $s.$ $r =$ $-EACCES|\!\}$
apply(*unfold smack-syslog-def*)
apply *wpsimp*
by (*metis* (*mono-tags*, *lifting*) *hoare-return-simp*)

lemma *smack-syslog-correctness3*:
$\exists s.$ $\{\!|\lambda s.$ *smack-privileged s CAP-MAC-OVERRIDE* $=$ *False*
$\wedge$ *smk-of-current s* $=$ *smack-syslog-label shared* $|\!\}$ *smack-syslog s t* $\{\!|\lambda r$ $s.$ $r =$ $0|\!\}$
apply(*unfold smack-syslog-def*)
apply *wpsimp*
by (*metis* (*mono-tags*, *lifting*) *hoare-return-simp*)

30.5 correctness lemmas of smack$_sb_alloc_security$

lemma *smack-sb-alloc-security-correctness*:
$\{\!|\lambda s.$ *True*$|\!\}$ *smack-sb-alloc-security s sb*
$\{\!|\lambda r$ $s.$ $(r = 0$ $)$ $\vee$ $r =$ (*uminus ENOMEM*) $|\!\}$
apply(*unfold smack-sb-alloc-security-def*)
apply *wpsimp*
done

30.6 correctness lemmas of smack$_sb_free_security$

lemma *smack-sb-free-security-correctness*:
$\{\!|\lambda s.$ *True*$|\!\}$ *smack-sb-free-security s sb* $\{\!|\lambda r$ $s.$ $r =$ *unit*$|\!\}$
apply(*unfold smack-sb-free-security-def get-sbnum-def*)
apply *wpsimp*
done

lemma *smack-sb-free-security-correctness1*:
$\{\!|\lambda s.$ *True*$|\!\}$ *smack-sb-free-security s sb* $\{\!|\lambda r$ $s.$ ($r =$ *unit* $\wedge$ *sb-security s sb* $=$ *None*)$|\!\}$
apply(*unfold smack-sb-free-security-def get-sbnum-def*)
apply *wpsimp*
done

30.7 correctness lemmas of smack$_sb_copy_data$

lemma *smack-sb-copy-data-correctness*:

$\{\!|\lambda s.\ True|\!\}$ *smack-sb-copy-data s orig smackopts*
$\{\!|\lambda r\ s.\ r = 0\ \lor r = (uminus\ ENOMEM)\ |\!\}$
apply(*unfold smack-sb-copy-data-def*)
apply *wpsimp*
done

30.8 correctness lemmas of smack$_{p}arse_{o}pts_{s}tr$

30.9 correctness lemmas of smack$_{s}et_{m}nt_{o}pts$

lemma *smack-set-mnt-opts-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-set-mnt-opts s sb opt kern-flags set-kern-flags*
$\{\!|\lambda r\ s.\ r = 0\ \lor r = (uminus\ EPERM)\ |\!\}$
apply(*unfold smack-set-mnt-opts-def*)
apply *wpsimp*
done

30.10 correctness lemmas of smack$_{s}b_{k}ern_{m}ount$

lemma *smack-sb-kern-mount-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-sb-kern-mount s sb f data* $\{\!|\lambda r\ s.\ (r = 0 \lor r = (uminus\ EPERM))|\!\}$
apply(*unfold smack-sb-kern-mount-def smack-set-mnt-opts-def*)
apply *wpsimp*
done

lemma *smack-sb-kern-mount-correctness1*:
$\exists\, data.\{\!|\lambda s.\ length(data) = 0|\!\}$ *smack-sb-kern-mount s sb f data* $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
apply(*unfold smack-sb-kern-mount-def smack-set-mnt-opts-def*)
apply *wpsimp*
by *blast*

30.11 correctness lemmas of smack$_{s}b_{s}tatfs$

lemma *smack-sb-statfs-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-sb-statfs s sb* $\{\!|\lambda r\ s.\ (r = 0 \lor r \neq 0)|\!\}$
apply(*unfold smack-sb-statfs-def*)
apply *wpsimp*
done

30.12 correctness lemmas of smack$_{b}prm_{s}et_{c}reds$

lemma *smack-bprm-set-creds-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-bprm-set-creds s bprm* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-bprm-set-creds-def smk-ptrace-rule-check-def ptrace-parent-def smack-privileged-cred-def*)
apply *wpsimp*
done

30.13 correctness lemmas of smack$_i node_a lloc_s ecurity$

lemma *smack-inode-alloc-security-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-alloc-security s a* $\{\!|\lambda r\ s.\ r = 0 \vee r = (uminus\ ENOMEM)$
$|\!\}$
apply(*unfold smack-inode-alloc-security-def*)
apply *wpsimp*
done

30.14 correctness lemmas of smack$_i node_f ree_s ecurity$

lemma *smack-inode-free-security-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-free-security s a* $\{\!|\lambda r\ s.\ r = unit\ |\!\}$
apply(*unfold smack-inode-free-security-def*)
apply *wpsimp*
done

30.15 correctness lemmas of smack$_i node_i nit_s ecurity$

lemma *smack-inode-init-security-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-init-security s inode dir qstr name value len'* $\{\!|\lambda r\ s.\ r = 0$
$\vee\ r \neq 0\ \ |\!\}$
apply *wpsimp*
done

30.16 correctness lemmas of smack$_i node_l ink$

lemma *smack-inode-link-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-link s old dir new* $\{\!|\lambda r\ s.\ r = 0 \vee r\neq 0|\!\}$
apply(*unfold smack-inode-link-def*)
apply *wpsimp*
done

30.17 correctness lemmas of smack$_i node_u nlink$

lemma *smack-inode-unlink-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-unlink s dir d* $\{\!|\lambda r\ s.\ r = 0 \vee r\neq 0|\!\}$
apply(*unfold smack-inode-unlink-def*)
apply *wpsimp*
done

30.18 correctness lemmas of smack$_i node_r mdir$

lemma *smack-inode-rmdir-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-rmdir s dir d* $\{\!|\lambda r\ s.\ r = 0 \vee r\neq 0|\!\}$
apply(*unfold smack-inode-rmdir-def*)
apply *wpsimp*
done

30.19 correctness lemmas of smack$_{i}$$node_{r}ename$

lemma *smack-inode-rename-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-rename s oldinode oldd newinode newd* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
apply(*unfold smack-inode-rename-def*)
apply *wpsimp*
done

30.20 correctness lemmas of smack$_{i}$$node_{p}ermission$

lemma *smack-inode-permission-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-permission s i mask′* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
apply(*unfold smack-inode-permission-def*)
apply *wpsimp*
done

30.21 correctness lemmas of smack$_{i}$$node_{s}etattr$

lemma *smack-inode-setattr-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-setattr s d attrs* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
apply(*unfold smack-inode-setattr-def*)
apply *wpsimp*
done

30.22 correctness lemmas of smack$_{i}$$node_{g}etattr$

lemma *smack-inode-getattr-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-getattr s path′* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
apply(*unfold smack-inode-getattr-def*)
apply *wpsimp*
done

30.23 correctness lemmas of smack$_{i}$$node_{s}etxattr$

lemma *smack-inode-setxattr-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-setxattr s dentry name value size′ flags′* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
apply(*unfold smack-inode-setxattr-def*)
apply *wpsimp*
done

30.24 correctness lemmas of smack$_{i}$$node_{p}ost_{s}etxattr$

lemma *smack-inode-post-setxattr-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-post-setxattr s dentry name value size′ flags′* $\{\!|\lambda r\ s.\ r = unit\ |\!\}$
apply(*unfold smack-inode-post-setxattr-def*)
apply *wpsimp*
done

30.25 correctness lemmas of smack$_i$$node_getxattr$

lemma *smack-inode-getxattr-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-getxattr s d name* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
apply(*unfold smack-inode-getxattr-def*)
apply *wpsimp*
done

30.26 correctness lemmas of smack$_i$$node_removexattr$

lemma *smack-inode-removexattr-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-removexattr s dentry name* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0|\!\}$
apply(*unfold smack-inode-removexattr-def*)
apply *wpsimp*
done

30.27 correctness lemmas of smack$_i$$node_getsecurity$

lemma *smack-inode-getsecurity-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-getsecurity s inode name buffer alloc* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq$
$0|\!\}$
apply(*unfold smack-inode-getsecurity-def*)
apply *wpsimp*
done

30.28 correctness lemmas of smack$_i$$node_listsecurity$

lemma *smack-inode-listsecurity-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-listsecurity s inode buffer buffer-size* $\{\!|\lambda r\ s.\ r = 17|\!\}$
apply(*unfold smack-inode-listsecurity-def*)
apply *wpsimp*
done

30.29 correctness lemmas of smack$_i$$node_getsecid$

lemma *smack-inode-getsecid-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-getsecid s i secid'* $\{\!|\lambda r\ s.\ r = unit|\!\}$
apply(*unfold smack-inode-getsecid-def*)
apply *wpsimp*
done

30.30 correctness lemmas of smack$_f$$ile_alloc_security$

lemma *smack-file-alloc-security-correctness*:
$\bigwedge s\ f.\ \{\!|\lambda s.\ True|\!\}$ *smack-file-alloc-security s f* $\{\!|\lambda r\ s.\ r = 0 \lor r = -EEXIST|\!\}$
apply(*unfold smack-file-alloc-security-def*)
apply *wpsimp*
done

lemma *smack-file-alloc-security-correctness-state*:

$\bigwedge s'\, f.\{\!|\lambda so.\ so = s' \wedge \textit{f-security so f} = \textit{None}|\!\}$
smack-file-alloc-security s' f
$\{\!|\lambda r\ sa.\ sa = s'(\!|\textit{f-security} := (\textit{f-security}\ s')(f := \textit{Some}\ (\textit{smk-of-current}\ s'))|\!)|\!\}$
apply(*unfold smack-file-alloc-security-def bind-def return-def modify-def put-def get-def EEXIST-def valid-def*)
apply *wpsimp*
done

30.31 correctness lemmas of smack$_f ile_f ree_s ecurity$

lemma *smack-file-free-security-correctness*:
$\{\!|\lambda s.\ \textit{True}|\!\}$ *smack-file-free-security s f* $\{\!|\lambda r\ s.\ r = \textit{unit}|\!\}$
apply(*unfold smack-file-free-security-def*)
apply *wpsimp*
done

30.32 correctness lemmas of smack$_m map_f ile$

lemma *smack-mmap-file-correctness*:
$\{\!|\lambda s.\ \textit{True}|\!\}$ *smack-mmap-file s file' reqprot prot flags'* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0|\!\}$
apply(*unfold smack-mmap-file-def*)
apply *wpsimp*
done

30.33 correctness lemmas of smack$_f ile_i octl$

lemma *smack-file-ioctl-correctness1*:
$\exists$ *file' inode* .
$\{\!|\lambda s.\ \textit{inode} = \textit{file-inode}(\textit{file}') \wedge \textit{unlikely}(\textit{IS-PRIVATE}(\textit{inode}))\ |\!\}$
smack-file-ioctl s file' cmd arg
$\{\!|\lambda r\ s.\ r = 0\ |\!\}$
apply(*unfold smack-file-ioctl-def*)
apply *wpsimp*
by (*smt hoare-assume-pre hoare-return-simp*)

lemma *smack-file-ioctl-correctness-ioc-write*:
$\exists$ *file' inode cmd ret*.
$\{\!|\lambda s.\ \textit{inode} = \textit{file-inode}(\textit{file}') \wedge$
$\textit{unlikely}(\textit{IS-PRIVATE}(\textit{inode})) = \textit{False} \wedge$
$\textit{cmd} = \textit{IOC-WRITE} \wedge$
$\textit{ret} = \textit{fst}(\textit{the-run-state}\ (\textit{smk-curacc}\ s\ (\textit{smk-of-inode}\ s\ \textit{inode})\ \textit{MAY-WRITE}\ \textit{ad})\ s)|\!\}$
smack-file-ioctl s file' cmd arg
$\{\!|\lambda r\ s.\ r = \textit{ret}\ |\!\}$
apply(*unfold smack-file-ioctl-def smk-bu-file-def bind-def return-def valid-def the-run-state-def fstI*)
by *auto*

lemma *smack-file-ioctl-correctness-ioc-read*:

$\exists$ *file' inode cmd ret.*
 $\{\!|\lambda s.$ *inode = file-inode(file')* $\wedge$
 unlikely(IS-PRIVATE(inode)) = False $\wedge$
 cmd = IOC-READ $\wedge$
 ret = fst(the-run-state (smk-curacc s (smk-of-inode s inode) MAY-READ ad) s)$|\!\}$
 smack-file-ioctl s file' cmd arg
 $\{\!|\lambda r\ s.\ r = ret\ |\!\}$
 apply(*unfold smack-file-ioctl-def smk-bu-file-def bind-def return-def valid-def the-run-state-def fstI*)
 by *auto*

lemma *smack-file-ioctl-correctness-ioc-other*:
$\exists$ *file' inode cmd.*
 $\{\!|\lambda s.$ *inode = file-inode(file')* $\wedge$
 unlikely(IS-PRIVATE(inode)) = False $\wedge$
 cmd $\neq$ *IOC-READ* $\wedge$
 cmd $\neq$ *IOC-WRITE*$|\!\}$
 smack-file-ioctl s file' cmd arg
 $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
 apply(*unfold smack-file-ioctl-def smk-bu-file-def bind-def return-def valid-def the-run-state-def fstI*)
 by *auto*

30.34 correctness lemmas of smack$_{f}ile_{l}ock$

lemma *smack-file-lock-correctness*:
$\{\!|\lambda s.$ *True*$|\!\}$ *smack-file-lock s file' cmd* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0|\!\}$
 apply(*unfold smack-file-lock-def*)
 apply *wpsimp*
 done

lemma *smack-file-lock-correctness1*:
$\exists$ *file' inode .*
 $\{\!|\lambda s.$ *inode = file-inode(file')* $\wedge$ *unlikely(IS-PRIVATE(inode))* $|\!\}$
 smack-file-lock s file' cmd
 $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
 apply(*unfold smack-file-lock-def*)
 apply *wpsimp*
 by (*smt hoare-assume-pre hoare-return-simp*)

lemma *smack-file-lock-correctness2*:
$\exists$ *file' inode rc.*
 $\{\!|\lambda s.$ *(SECURITY-SMACK-BRINGUP conf) = True* $\wedge$ *inode = file-inode(file')* $\wedge$ *unlikely(IS-PRIVATE(inode)) = False* $|\!\}$
 smack-file-lock s file' cmd
 $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
 apply(*unfold smack-file-lock-def smk-bu-file-def bind-def return-def valid-def*)
 by *auto*

lemma *smack-file-lock-correctness3*:
$\forall sa.\ \exists file'\ inode\ ret\ ad.$
$\{\lambda s.\ (SECURITY\text{-}SMACK\text{-}BRINGUP\ conf) = False \wedge s = sa \wedge$
$inode = file\text{-}inode(file') \wedge$
$unlikely(IS\text{-}PRIVATE(inode)) = False \wedge$
$ret = fst(the\text{-}run\text{-}state\ (smk\text{-}curacc\ s\ (smk\text{-}of\text{-}inode\ s\ inode)\ MAY\text{-}LOCK\ ad)\ s)\}$
$smack\text{-}file\text{-}lock\ sa\ file'\ cmd$
$\{\lambda r\ s.\ r = ret\ \}$
apply(*unfold smack-file-lock-def smk-bu-file-def bind-def return-def valid-def the-run-state-def fstI*)
apply *auto*
by *blast*

30.35 correctness lemmas of smack$_{f}ile_{f}cntl_{d}ef$

lemma *smack-file-fcntl-correctness*:
$\{\lambda s.\ True\}\ smack\text{-}file\text{-}fcntl\ s\ file'\ cmd\ arg\ \{\lambda r\ s.\ r = 0 \vee r \neq 0\}$
apply(*unfold smack-file-fcntl-def*)
apply *wpsimp*
done

lemma *smack-file-fcntl-correctness1*:
$\exists file'\ inode\ .$
$\{\lambda s.\ inode = file\text{-}inode(file') \wedge unlikely(IS\text{-}PRIVATE(inode))\ \}$
$smack\text{-}file\text{-}fcntl\ s\ file'\ cmd\ arg$
$\{\lambda r\ s.\ r = 0\ \}$
apply(*unfold smack-file-fcntl-def*)
apply *wpsimp*
by (*smt hoare-assume-pre hoare-return-simp*)

lemma *smack-file-fcntl-correctness-fsetlkandfsetlkw*:
$\forall sa.\ \exists file'\ inode\ ret\ ad\ cmd.$
$\{\lambda s.\ (SECURITY\text{-}SMACK\text{-}BRINGUP\ conf) = False \wedge s = sa \wedge$
$inode = file\text{-}inode(file') \wedge$
$unlikely(IS\text{-}PRIVATE(inode)) = False \wedge$
$(cmd = F\text{-}SETLK \vee cmd = F\text{-}SETLKW) \wedge$
$ret = fst(the\text{-}run\text{-}state\ (smk\text{-}curacc\ s\ (smk\text{-}of\text{-}inode\ s\ inode)\ MAY\text{-}LOCK\ ad)\ s)\}$
$smack\text{-}file\text{-}fcntl\ s\ file'\ cmd\ arg$
$\{\lambda r\ s.\ r = ret\ \}$
apply(*unfold smack-file-fcntl-def smk-bu-file-def bind-def return-def valid-def the-run-state-def fstI*)
apply *auto*
by *blast*

lemma *smack-file-fcntl-correctness-fsetownandsig*:
$\forall sa\ .\exists file'\ inode\ ret\ ad\ \ cmd.$

$\{\!|\lambda s.$ (*SECURITY-SMACK-BRINGUP conf*) = *False* $\wedge$ *s* = *sa* $\wedge$
inode = *file-inode*(*file'*) $\wedge$
unlikely(*IS-PRIVATE*(*inode*)) = *False* $\wedge$
(*cmd* = *F-SETOWN* $\vee$ *cmd* = *F-SETSIG*) $\wedge$
ret = *fst*(*the-run-state* (*smk-curacc s* (*smk-of-inode s inode*) *MAY-WRITE* *ad*) *s*)$|\!\}$
smack-file-fcntl s file' cmd arg
$\{\!|\lambda r\ s.\ r = ret\ |\!\}$
apply(*unfold smack-file-fcntl-def smk-bu-file-def bind-def return-def valid-def the-run-state-def fstI*)
apply *auto*
by *blast*

lemma *smack-file-fcntl-correctness-other*:
$\forall$ *sa*. $\exists$ *file' inode ret ad cmd*.
$\{\!|\lambda s.$ (*SECURITY-SMACK-BRINGUP conf*) = *False* $\wedge$ *s* = *sa* $\wedge$
inode = *file-inode*(*file'*) $\wedge$
unlikely(*IS-PRIVATE*(*inode*)) = *False* $\wedge$
(*cmd* $\neq$ *F-SETOWN* $\wedge$ *cmd* $\neq$ *F-SETSIG* $\wedge$*cmd* $\neq$ *F-SETLK* $\wedge$ *cmd* $\neq$ *F-SETLKW*) $\wedge$
ret = *fst*(*the-run-state* (*smk-curacc s* (*smk-of-inode s inode*) *MAY-WRITE* *ad*) *s*)$|\!\}$
smack-file-fcntl s file' cmd arg
$\{\!|\lambda r\ s.\ r = 0\ |\!\}$
apply(*unfold smack-file-fcntl-def smk-bu-file-def bind-def return-def valid-def the-run-state-def fstI*)
apply *auto*
done

30.36 correctness lemmas of smack$_{f}ile_{s}et_{f}owner$

lemma *smack-file-set-fowner-correctness*:
$\{\!|\lambda s.$ *True*$|\!\}$ *smack-file-set-fowner s file'* $\{\!|\lambda r\ s.\ r = unit\ |\!\}$
apply(*unfold smack-file-set-fowner-def*)
apply *wpsimp*
done

lemma *smack-file-set-fowner-correctness1*:
$\bigwedge$*sa*. $\{\!|\lambda s.\ s = sa|\!\}$ *smack-file-set-fowner sa file'* $\{\!|\lambda r\ s.$ *f-security s file'* = *Some* (*smk-of-current sa*) $|\!\}$
apply(*unfold smack-file-set-fowner-def modify-def return-def get-def put-def bind-def valid-def*)
apply *wpsimp*
done

30.37 correctness lemmas of smack$_{f}ile_{s}end_{s}igiotask$

lemma *smack-file-send-sigiotask-correctness*:
$\{\!|\lambda s.$ *True*$|\!\}$ *smack-file-send-sigiotask s tsk' fown signum* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0|\!\}$
apply(*unfold smack-file-send-sigiotask-def*)

apply *wpsimp*
done

30.38 correctness lemmas of smack$_{f}ile_{r}eceive$

lemma *smack-file-receive-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-file-receive s file'* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0|\!\}$
apply(*unfold smack-file-receive-def*)
apply *wpsimp*
done

30.39 correctness lemmas of smack$_{f}ile_{o}pen$

lemma *smack-file-open-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-file-open s file'* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0|\!\}$
apply(*unfold smack-file-open-def*)
apply *wpsimp*
done

30.40 correctness lemmas of smack$_{c}red_{a}lloc_{b}lank$

lemma *smack-cred-alloc-blank-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-cred-alloc-blank s c g* $\{\!|\lambda r\ s.\ r = 0 \vee r = -ENOMEM\ |\!\}$
apply(*unfold smack-cred-alloc-blank-def*)
apply *wpsimp*
done

30.41 correctness lemmas of smack$_{c}red_{f}ree$

lemma *smack-cred-free-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-cred-free s c* $\{\!|\lambda r\ s.\ r = unit \wedge (t\text{-}security\ s)\ c = None\ |\!\}$
apply(*unfold smack-cred-free-def*)
apply *wpsimp*
done

30.42 correctness lemmas of smack$_{c}red_{p}repare$

lemma *smack-cred-prepare-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-cred-prepare s new old g* $\{\!|\lambda r\ s.\ r = 0 \vee r \neq 0\ |\!\}$
apply(*unfold smack-cred-prepare-def*)
apply *wpsimp*
done

30.43 correctness lemmas of smack$_{c}red_{g}etsecid$

lemma *smack-cred-getsecid-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-cred-getsecid s c i* $\{\!|\lambda r\ s'.\ r = unit\ |\!\}$
apply(*unfold smack-cred-getsecid-def smk-of-task-def*)
apply *wpsimp*
done

30.44 correctness lemmas of smack$_c red_t ransfer$

lemma *smack-cred-transfer-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-cred-transfer s new old* $\{\!|\lambda r\ s'.\ r = unit\ |\!\}$
apply(*unfold smack-cred-transfer-def smk-of-task-def*)
apply *wpsimp*
done

30.45 correctness lemmas of smack$_k ernel_a ct_a s$

lemma *smack-kernel-act-as-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-kernel-act-as s c i* $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
apply(*unfold smack-kernel-act-as-def*)
apply *wpsimp*
done

30.46 correctness lemmas of smack$_k ernel_c reate_f iles_a s$

lemma *smack-kernel-create-files-as-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-kernel-create-files-as s new i* $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
apply(*unfold smack-kernel-create-files-as-def*)
apply *wpsimp*
done

30.47 correctness lemmas of smk$_c uracc_o n_t ask$

lemma *smk-curacc-on-task-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smk-curacc-on-task s p access' caller'* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smk-curacc-on-task-def*)
apply *wpsimp*
done

30.48 correctness lemmas of smack$_t ask_s etpgid$

lemma *smack-task-setpgid-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-task-setpgid s p pid* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-task-setpgid-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.49 correctness lemmas of smack$_t ask_g etpgid$

lemma *smack-task-getpgid-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-task-getpgid s p* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-task-getpgid-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.50 correctness lemmas of smack$_t ask_g etsid$

lemma *smack-task-getsid-correctness*:

$\{\lambda s.\ True\}$ *smack-task-getsid s p* $\{\lambda r\ s.\ r = 0 \lor r \neq 0\ \}$
apply(*unfold smack-task-getsid-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.51 correctness lemmas of smack$_{t}ask_{g}etsecid$

lemma *smack-task-getsecid-correctness*:
$\{\lambda s.\ True\}$ *smack-task-getsecid s p pid* $\{\lambda r\ s.\ r = unit\ \}$
apply(*unfold smack-task-getsecid-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.52 correctness lemmas of smack$_{t}ask_{s}etnice$

lemma *smack-task-setnice-correctness*:
$\{\lambda s.\ True\}$ *smack-task-setnice s p pid* $\{\lambda r\ s.\ r = 0 \lor r \neq 0\ \}$
apply(*unfold smack-task-setnice-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.53 correctness lemmas of smack$_{t}ask_{s}etioprio$

lemma *smack-task-setioprio-correctness*:
$\{\lambda s.\ True\}$ *smack-task-setioprio s p pid* $\{\lambda r\ s.\ r = 0 \lor r \neq 0\ \}$
apply(*unfold smack-task-setioprio-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.54 correctness lemmas of smack$_{t}ask_{g}etioprio$

lemma *smack-task-getioprio-correctness*:
$\{\lambda s.\ True\}$ *smack-task-getioprio s p* $\{\lambda r\ s.\ r = 0 \lor r \neq 0\ \}$
apply(*unfold smack-task-getioprio-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.55 correctness lemmas of smack$_{t}ask_{s}etscheduler$

lemma *smack-task-setscheduler-correctness*:
$\{\lambda s.\ True\}$ *smack-task-setscheduler s p* $\{\lambda r\ s.\ r = 0 \lor r \neq 0\ \}$
apply(*unfold smack-task-setscheduler-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.56 correctness lemmas of smack$_{t}ask_{g}etscheduler$

lemma *smack-task-getscheduler-correctness*:
$\{\lambda s.\ True\}$ *smack-task-getscheduler s p* $\{\lambda r\ s.\ r = 0 \lor r \neq 0\ \}$
apply(*unfold smack-task-getscheduler-def smk-curacc-on-task-def*)

apply *wpsimp*
done

30.57 correctness lemmas of smack$_{t}ask_{m}ovememory$

lemma *smack-task-movememory-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-task-movememory s p* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-task-movememory-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.58 correctness lemmas of smack$_{t}ask_{k}ill$

lemma *smack-task-kill-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-task-kill s p info sig c* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-task-kill-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.59 correctness lemmas of smack$_{t}ask_{t}o_{i}node$

lemma *smack-task-to-inode-correctness*:
$\forall p.\ \{\!|\lambda s.\ True|\!\}$ *smack-task-to-inode s p i* $\{\!|\lambda r\ s'.\ r = unit\ |\!\}$
apply(*unfold smack-task-to-inode-def smk-curacc-on-task-def*)
apply *wpsimp*
done

30.60 correctness lemmas of smack$_{i}pc_{p}ermission$

lemma *smack-ipc-permission-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-ipc-permission s ipp flag* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-ipc-permission-def*)
apply *wpsimp*
done

30.61 correctness lemmas of smack$_{i}pc_{g}etsecid$

lemma *smack-ipc-getsecid-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-ipc-getsecid s ipp flag* $\{\!|\lambda r\ s'.\ r = ()\ |\!\}$
apply(*unfold smack-ipc-getsecid-def*)
apply *wpsimp*
done

30.62 correctness lemmas of smack$_{m}sg_{m}sg_{a}lloc_{s}ecurity$

lemma *smack-msg-msg-alloc-security-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-msg-msg-alloc-security s msg* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-msg-msg-alloc-security-def*)
apply *wpsimp*
done

lemma *smack-msg-msg-alloc-security-correctness-state*:
$\bigwedge sa\ msg.\ \{\!|\lambda s\ .s = sa \wedge msg\text{-}security\ s\ msg = None|\!\}$
$smack\text{-}msg\text{-}msg\text{-}alloc\text{-}security\ sa\ msg$
$\{\!|\lambda r\ s.\ msg\text{-}security\ s\ msg = Some\ (smk\text{-}of\text{-}current\ sa)|\!\}$
apply(*unfold smack-msg-msg-alloc-security-def*)
apply *wpsimp*
done

30.63 correctness lemmas of smack$_{m}sg_{m}sg_{f}ree_{s}ecurity$

lemma *smack-msg-msg-free-security-correctness*:
$\{\!|\lambda s.\ True|\!\}\ smack\text{-}msg\text{-}msg\text{-}free\text{-}security\ s\ msg\ \ \{\!|\lambda r\ s'.\ r = ()\ |\!\}$
apply(*unfold smack-msg-msg-free-security-def*)
apply *wpsimp*
done

30.64 correctness lemmas of smack$_{i}pc_{a}lloc_{s}ecurity$

lemma *smack-ipc-alloc-security-correctness*:
$\{\!|\lambda s.\ True|\!\}\ smack\text{-}ipc\text{-}alloc\text{-}security\ s\ isp\ \ \{\!|\lambda r\ s.\ r = 0\ |\!\}$
apply(*unfold smack-ipc-alloc-security-def*)
apply *wpsimp*
done

30.65 correctness lemmas of smack$_{i}pc_{f}ree_{s}ecurity$

lemma *smack-ipc-free-security-correctness*:
$\{\!|\lambda s.\ True|\!\}\ smack\text{-}ipc\text{-}free\text{-}security\ s\ isp\ \ \{\!|\lambda r\ s.\ r = ()\ |\!\}$
apply(*unfold smack-ipc-free-security-def*)
apply *wpsimp*
done

30.66 correctness lemmas of smack$_{s}hm_{a}ssociate$

lemma *smack-shm-associate-correctness*:
$\{\!|\lambda s.\ True|\!\}\ smack\text{-}shm\text{-}associate\ s\ isp\ shmflg\ \ \{\!|\lambda r\ s'.\ r = 0 \vee r \neq 0\ |\!\}$
apply(*unfold smack-shm-associate-def*)
apply *wpsimp*
done

30.67 correctness lemmas of smack$_{s}hm_{s}hmctl$

lemma *smack-shm-shmctl-correctness*:
$\{\!|\lambda s.\ True|\!\}\ smack\text{-}shm\text{-}shmctl\ s\ isp\ cmd\ \ \ \{\!|\lambda r\ s.\ r = 0 \vee r \neq 0\ |\!\}$
apply(*unfold smack-shm-shmctl-def*)
apply *wpsimp*
done

30.68 correctness lemmas of smack$_shm_shmat$

lemma *smack-shm-shmat-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-shm-shmat s ipc' shmaddr shmflg* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-shm-shmat-def*)
apply *wpsimp*
done

30.69 correctness lemmas of smk$_curacc_sem$

lemma *smk-curacc-sem-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smk-curacc-sem s isp access* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smk-curacc-sem-def*)
apply *wpsimp*
done

30.70 correctness lemmas of smack$_sem_associate$

lemma *smack-sem-associate-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-sem-associate s isp shmflg* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-sem-associate-def*)
apply *wpsimp*
done

30.71 correctness lemmas of smack$_sem_semctl$

lemma *smack-sem-semctl-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-sem-semctl s isp cmd* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-sem-semctl-def*)
apply *wpsimp*
done

30.72 correctness lemmas of smack$_sem_semop$

lemma *smack-sem-semop-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-sem-semop s isp sops nsops alter* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-sem-semop-def*)
apply *wpsimp*
done

30.73 correctness lemmas of smk$_curacc_msq$

lemma *smk-curacc-msq-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smk-curacc-msq s isp acces* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smk-curacc-msq-def*)
apply *wpsimp*
done

30.74 correctness lemmas of smack$_msg_queue_associate$

lemma *smack-msg-associate-correctness*:

$\{\!|\lambda s.\ True|\!\}$ *smack-msg-queue-associate s isp shmflg* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
  apply(*unfold smack-msg-queue-associate-def*)
  apply *wpsimp*
  done

30.75 correctness lemmas of smack$_msg_queue_msgctl$

lemma *smack-msg-queue-msgctl-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-msg-queue-msgctl s isp cmd* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
  apply(*unfold smack-msg-queue-msgctl-def*)
  apply *wpsimp*
  done

30.76 correctness lemmas of smack$_msg_queue_msgsnd$

lemma *smack-msg-queue-msgsnd-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-msg-queue-msgsnd s isp msg msqflg* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
  apply(*unfold smack-msg-queue-msgsnd-def*)
  apply *wpsimp*
  done

30.77 correctness lemmas of smack$_msg_queue_msgrcv$

lemma *smack-msg-queue-msgrcv-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-msg-queue-msgrcv s isp msg p long msqflg* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
  apply(*unfold smack-msg-queue-msgrcv-def*)
  apply *wpsimp*
  done

30.78 correctness lemmas of smack$_key_alloc$

lemma *smack-key-alloc-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-key-alloc s k c flg* $\{\!|\lambda r\ s.\ r = 0\ |\!\}$
  apply(*unfold smack-key-alloc-def*)
  apply *wpsimp*
  done

30.79 correctness lemmas of smack$_key_free$

lemma *smack-key-free-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-key-free s k* $\{\!|\lambda r\ s.\ r = unit\ |\!\}$
  apply(*unfold smack-key-free-def*)
  apply *wpsimp*
  done

30.80 correctness lemmas of smack$_key_permission$

lemma *smack-key-permission-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-key-permission s key-ref c perm* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$

apply(*unfold smack-key-permission-def*)
apply *wpsimp*
done

30.81 correctness lemmas of smack$_k ey_g etsecurity$

lemma *smack-key-getsecurity-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-key-getsecurity s k buffer* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-key-getsecurity-def*)
apply *wpsimp*
done

30.82 correctness lemmas of smack$_u nix_s tream_c onnect$

lemma *smack-unix-stream-connect-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-unix-stream-connect s sock other newsk* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-unix-stream-connect-def*)
apply *wpsimp*
done

30.83 correctness lemmas of smack$_u nix_m ay_s end$

lemma *smack-unix-may-send-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-unix-may-send s sock other* $\{\!|\lambda r\ s.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-unix-may-send-def*)
apply *wpsimp*
done

30.84 correctness lemmas of smack$_s ocket_p ost_c reate$

lemma *smack-socket-post-create-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-socket-post-create s sock family type′ protocols kern* $\{\!|\lambda r\ s'.\ r =$
$0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-socket-post-create-def*)
apply *wpsimp*
done

30.85 correctness lemmas of smack$_s ocket_s ocketpair$

lemma *smack-socket-socketpair-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-socket-socketpair s socka sockb* $\{\!|\lambda r\ s'.\ r = 0\ |\!\}$
apply(*unfold smack-socket-socketpair-def*)
apply *wpsimp*
done

30.86 correctness lemmas of smack$_s ocket_b ind$

lemma *smack-socket-bind-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-socket-bind s sock address addrlen* $\{\!|\lambda r\ s'.\ r = 0\ |\!\}$
apply(*unfold smack-socket-bind-def*)

apply *wpsimp*
done

30.87 correctness lemmas of smack$_{s}ocket_{c}onnect$

lemma *smack-socket-connect-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-socket-connect s sock sap addrlen* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-socket-connect-def*)
apply *wpsimp*
done

30.88 correctness lemmas of smack$_{s}ocket_{s}endmsg$

lemma *smack-socket-sendmsg-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-socket-sendmsg s sock msg size'* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-socket-sendmsg-def*)
apply *wpsimp*
done

30.89 correctness lemmas of smack$_{s}ocket_{s}ock_{r}cv_{s}kb$

lemma *smack-socket-sock-rcv-skb-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-socket-sock-rcv-skb s sock skb* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-socket-sock-rcv-skb-def*)
apply *wpsimp*
done

30.90 correctness lemmas of smack$_{s}ocket_{g}etpeersec_{s}tream$

lemma *smack-socket-getpeersec-stream-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-socket-getpeersec-stream s sock optval optlen len'* $\{\!|\lambda r\ s'.\ r = 0$
$\lor\ r \neq 0\ |\!\}$
apply(*unfold smack-socket-getpeersec-stream-def*)
apply *wpsimp*
done

30.91 correctness lemmas of smack$_{s}ocket_{g}etpeersec_{d}gram$

lemma *smack-socket-getpeersec-dgram-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-socket-getpeersec-dgram s sock skb secid'* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq$
$0\ |\!\}$
apply(*unfold smack-socket-getpeersec-dgram-def*)
apply *wpsimp*
done

30.92 correctness lemmas of smack$_{s}k_{a}lloc_{s}ecurity$

lemma *smack-sk-alloc-security-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-sk-alloc-security s sock family flgs* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
apply(*unfold smack-sk-alloc-security-def*)

apply *wpsimp*
done

30.93 correctness lemmas of smack$_s k_f ree_s ecurity$

lemma *smack-sk-free-security-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-sk-free-security s sock* $\{\!|\lambda r\ s'.\ r = unit\ |\!\}$
apply(*unfold smack-sk-free-security-def*)
apply *wpsimp*
done

30.94 correctness lemmas of smack$_s ock_g raft$

lemma *smack-sock-graft-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-sock-graft s sock parent'* $\{\!|\lambda r\ s'.\ r = unit\ |\!\}$
apply(*unfold smack-sock-graft-def*)
apply *wpsimp*
done

30.95 correctness lemmas of smack$_i net_c onn_r equest$

lemma *smack-inet-conn-request-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inet-conn-request s sock skb req* $\{\!|\lambda r\ s'.\ r = 0 \vee r \neq 0\ |\!\}$
apply(*unfold smack-inet-conn-request-def*)
apply *wpsimp*
done

30.96 correctness lemmas of smack$_i net_c sk_c lone$

lemma *smack-inet-csk-clone-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inet-csk-clone s sock req* $\{\!|\lambda r\ s'.\ r = unit\ |\!\}$
apply(*unfold smack-inet-csk-clone-def*)
apply *wpsimp*
done

30.97 correctness lemmas of smack$_a udit_r ule_i nit$

lemma *smack-audit-rule-init-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-audit-rule-init s field op rulestr vrule* $\{\!|\lambda r\ s'.\ r = 0 \vee r \neq 0\ |\!\}$
apply *wpsimp*
done

30.98 correctness lemmas of smack$_a udit_r ule_k nown$

lemma *smack-audit-rule-known-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-audit-rule-known s krule* $\{\!|\lambda r\ s'.\ r = 0 \vee r \neq 0\ |\!\}$
apply *wpsimp*
done

30.99 correctness lemmas of smack$_a udit_r ule_m atch$

lemma *smack-audit-rule-match-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-audit-rule-match s secid′ field op vrule actx* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
  apply *wpsimp*
  done

30.100 correctness lemmas of smack$_i smaclabel$

lemma *smack-ismaclabel-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-ismaclabel s name* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
  apply *wpsimp*
  done

30.101 correctness lemmas of smack$_s ecid_t o_s ecctx$

lemma *smack-secid-to-secctx-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-secid-to-secctx s secid′ secdata seclen* $\{\!|\lambda r\ s.\ r = 0|\!\}$
  apply(*unfold smack-secid-to-secctx-def*)
  apply *wpsimp*
  done

30.102 correctness lemmas of smack$_s ecctx_t o_s ecid$

lemma *smack-secctx-to-secid-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-secctx-to-secid s secdata seclen secid′* $\{\!|\lambda r\ s.\ r = 0|\!\}$
  apply(*unfold smack-secctx-to-secid-def*)
  apply *wpsimp*
  done

30.103 correctness lemmas of smack$_i node_n otifysecctx$

lemma *smack-inode-notifysecctx-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-notifysecctx s inode ctx ctxlen* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
  apply(*unfold smack-inode-notifysecctx-def*)
  apply *wpsimp*
  done

30.104 correctness lemmas of smack$_i node_s etsecctx$

lemma *smack-inode-setsecctx-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-setsecctx s dentry ctx ctxlen* $\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
  apply(*unfold smack-inode-setsecctx-def*)
  apply *wpsimp*
  done

30.105 correctness lemmas of smack$_i node_g etsecctx$

lemma *smack-inode-getsecctx-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-getsecctx s inode ctx ctxlen* $\{\!|\lambda r\ s.\ r = 0|\!\}$

apply(*unfold smack-inode-getsecctx-def*)
apply *wpsimp*
done

30.106 correctness lemmas of smack$_{i}node_{c}opy_{u}p$

lemma *smack-inode-copy-up-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-copy-up s dentry new*$\{\!|\lambda r\ s'.\ r = 0 \lor r \neq 0\ |\!\}$
apply *wpsimp*
done

30.107 correctness lemmas of smack$_{i}node_{c}opy_{u}p_{x}attr$

lemma *smack-inode-copy-up-xattr-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-inode-copy-up-xattr s name* $\{\!|\lambda r\ s.\ r = -EOPNOTSUPP \lor r = 1|\!\}$
apply(*unfold smack-inode-copy-up-xattr-def*)
apply *wpsimp*
done

lemma *smack-inode-copy-up-xattr-correctness1*:
$\{\!|\lambda s.\ name = XATTR\text{-}NAME\text{-}SMACK|\!\}$ *smack-inode-copy-up-xattr s name* $\{\!|\lambda r\ s.\ r = 1|\!\}$
apply(*unfold smack-inode-copy-up-xattr-def*)
apply *wpsimp*
done

lemma *smack-inode-copy-up-xattr-correctness2*:
$\{\!|\lambda s.\ name \neq\ XATTR\text{-}NAME\text{-}SMACK|\!\}$ *smack-inode-copy-up-xattr s name* $\{\!|\lambda r\ s.\ r = -EOPNOTSUPP|\!\}$
apply(*unfold smack-inode-copy-up-xattr-def*)
apply *wpsimp*
done

30.108 correctness lemmas of smack$_{d}entry_{c}reate_{f}iles_{a}s$

lemma *smack-dentry-create-files-as-correctness*:
$\{\!|\lambda s.\ True|\!\}$ *smack-dentry-create-files-as s dentry mode' name old new* $\{\!|\lambda r\ s.\ r = 0|\!\}$
apply(*unfold smack-dentry-create-files-as-def*)
apply *wpsimp*
done
end