Hoare Logic for Parallel Programs

Leonor Prensa Nieto

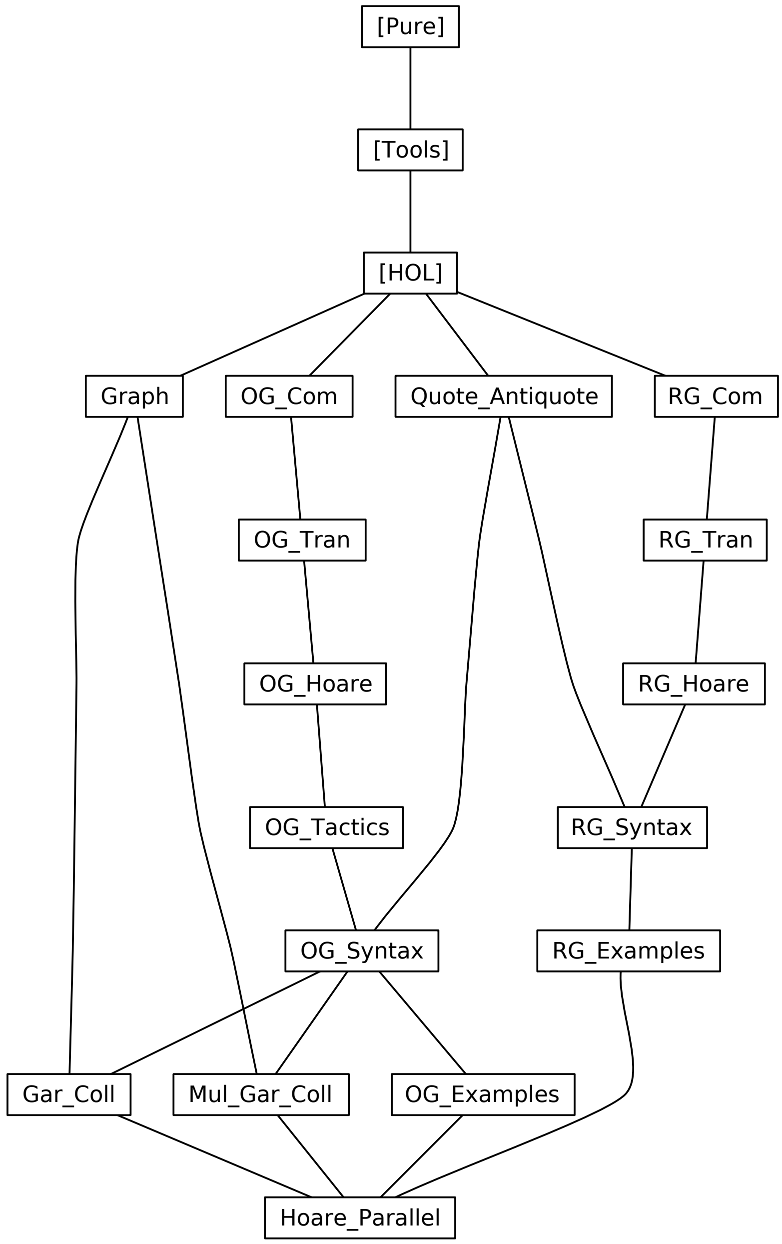
April 15, 2020

Abstract

In the following theories a formalization of the Owicki-Gries and the rely-guarantee methods is presented. These methods are widely used for correctness proofs of parallel imperative programs with shared variables. We define syntax, semantics and proof rules in Isabelle/HOL. The proof rules also provide for programs parameterized in the number of parallel components. Their correctness w.r.t. the semantics is proven. Completeness proofs for both methods are extended to the new case of parameterized programs. (These proofs have not been formalized in Isabelle. They can be found in [1].) Using this formalizations we verify several non-trivial examples for parameterized and non-parameterized programs. For the automatic generation of verification conditions with the Owicki-Gries method we define a tactic based on the proof rules. The most involved examples are the verification of two garbage-collection algorithms, the second one parameterized in the number of mutators.

For excellent descriptions of this work see [2, 4, 1, 3].

Contents


[Pure]
[Tools]
[HOL]
Graph
OG_Com
Quote_Antiquote
RG_Com
OG_Tran
RG_Tran
OG_Hoare
RG_Hoare
OG_Tactics
RG_Syntax
OG_Syntax
RG_Examples
Gar_Coll
Mul_Gar_Coll
OG_Examples
Hoare_Parallel

Chapter 1

The Owicki-Gries Method

1.1 Abstract Syntax

theory *OG-Com* **imports** *Main* **begin**

Type abbreviations for boolean expressions and assertions:

type-synonym $'a\ bexp = 'a\ set$
type-synonym $'a\ assn = 'a\ set$

The syntax of commands is defined by two mutually recursive datatypes: *'a ann-com* for annotated commands and *'a com* for non-annotated commands.

datatype $'a$ *ann-com* =
  AnnBasic ($'a$ *assn*) ($'a \Rightarrow 'a$)
  | *AnnSeq* ($'a$ *ann-com*) ($'a$ *ann-com*)
  | *AnnCond1* ($'a$ *assn*) ($'a$ *bexp*) ($'a$ *ann-com*) ($'a$ *ann-com*)
  | *AnnCond2* ($'a$ *assn*) ($'a$ *bexp*) ($'a$ *ann-com*)
  | *AnnWhile* ($'a$ *assn*) ($'a$ *bexp*) ($'a$ *assn*) ($'a$ *ann-com*)
  | *AnnAwait* ($'a$ *assn*) ($'a$ *bexp*) ($'a$ *com*)
and $'a$ *com* =
  Parallel ($'a$ *ann-com option* $\times$ $'a$ *assn*) *list*
  | *Basic* ($'a \Rightarrow 'a$)
  | *Seq* ($'a$ *com*) ($'a$ *com*)
  | *Cond* ($'a$ *bexp*) ($'a$ *com*) ($'a$ *com*)
  | *While* ($'a$ *bexp*) ($'a$ *assn*) ($'a$ *com*)

The function *pre* extracts the precondition of an annotated command:

primrec *pre* ::$'a$ *ann-com* $\Rightarrow$ $'a$ *assn* **where**
  pre (*AnnBasic r f*) = *r*
| *pre* (*AnnSeq c1 c2*) = *pre c1*
| *pre* (*AnnCond1 r b c1 c2*) = *r*
| *pre* (*AnnCond2 r b c*) = *r*
| *pre* (*AnnWhile r b i c*) = *r*
| *pre* (*AnnAwait r b c*) = *r*

Well-formedness predicate for atomic programs:

primrec *atom-com* :: *'a com* ⇒ *bool* **where**
atom-com (*Parallel Ts*) = *False*
| *atom-com* (*Basic f*) = *True*
| *atom-com* (*Seq c1 c2*) = (*atom-com c1* ∧ *atom-com c2*)
| *atom-com* (*Cond b c1 c2*) = (*atom-com c1* ∧ *atom-com c2*)
| *atom-com* (*While b i c*) = *atom-com c*

end

1.2 Operational Semantics

theory *OG-Tran* **imports** *OG-Com* **begin**

type-synonym *'a ann-com-op* = (*'a ann-com*) *option*
type-synonym *'a ann-triple-op* = (*'a ann-com-op* × *'a assn*)

primrec *com* :: *'a ann-triple-op* ⇒ *'a ann-com-op* **where**
com (*c*, *q*) = *c*

primrec *post* :: *'a ann-triple-op* ⇒ *'a assn* **where**
post (*c*, *q*) = *q*

definition *All-None* :: *'a ann-triple-op list* ⇒ *bool* **where**
All-None Ts ≡ ∀ (*c*, *q*) ∈ *set Ts*. *c* = *None*

1.2.1 The Transition Relation

inductive-set
ann-transition :: ((*'a ann-com-op* × *'a*) × (*'a ann-com-op* × *'a*)) *set*
and *transition* :: ((*'a com* × *'a*) × (*'a com* × *'a*)) *set*
and *ann-transition'* :: (*'a ann-com-op* × *'a*) ⇒ (*'a ann-com-op* × *'a*) ⇒ *bool*
(- −*1*→ -[*81*,*81*] *100*)
and *transition'* :: (*'a com* × *'a*) ⇒ (*'a com* × *'a*) ⇒ *bool*
(- −*P1*→ -[*81*,*81*] *100*)
and *transitions* :: (*'a com* × *'a*) ⇒ (*'a com* × *'a*) ⇒ *bool*
(- −*P*∗→ -[*81*,*81*] *100*)
where
con-0 −*1*→ *con-1* ≡ (*con-0*, *con-1*) ∈ *ann-transition*
| *con-0* −*P1*→ *con-1* ≡ (*con-0*, *con-1*) ∈ *transition*
| *con-0* −*P*∗→ *con-1* ≡ (*con-0*, *con-1*) ∈ *transition**

| *AnnBasic*: (*Some* (*AnnBasic r f*), *s*) −*1*→ (*None*, *f s*)

| *AnnSeq1*: (*Some c0*, *s*) −*1*→ (*None*, *t*) ⟹
(*Some* (*AnnSeq c0 c1*), *s*) −*1*→ (*Some c1*, *t*)
| *AnnSeq2*: (*Some c0*, *s*) −*1*→ (*Some c2*, *t*) ⟹
(*Some* (*AnnSeq c0 c1*), *s*) −*1*→ (*Some* (*AnnSeq c2 c1*), *t*)

| *AnnCond1T*: *s* ∈ *b* ⟹ (*Some* (*AnnCond1 r b c1 c2*), *s*) −*1*→ (*Some c1*, *s*)

| *AnnCond1F*: $s \notin b \Longrightarrow (Some\ (AnnCond1\ r\ b\ c1\ c2),\ s)\ {-1\rightarrow}\ (Some\ c2,\ s)$

| *AnnCond2T*: $s \in b \Longrightarrow (Some\ (AnnCond2\ r\ b\ c),\ s)\ {-1\rightarrow}\ (Some\ c,\ s)$
| *AnnCond2F*: $s \notin b \Longrightarrow (Some\ (AnnCond2\ r\ b\ c),\ s)\ {-1\rightarrow}\ (None,\ s)$

| *AnnWhileF*: $s \notin b \Longrightarrow (Some\ (AnnWhile\ r\ b\ i\ c),\ s)\ {-1\rightarrow}\ (None,\ s)$
| *AnnWhileT*: $s \in b \Longrightarrow (Some\ (AnnWhile\ r\ b\ i\ c),\ s)\ {-1\rightarrow}$
$(Some\ (AnnSeq\ c\ (AnnWhile\ i\ b\ i\ c)),\ s)$

| *AnnAwait*: $[\![\ s \in b;\ atom\text{-}com\ c;\ (c,\ s)\ {-P{*}\rightarrow}\ (Parallel\ [],\ t)\ ]\!] \Longrightarrow$
$(Some\ (AnnAwait\ r\ b\ c),\ s)\ {-1\rightarrow}\ (None,\ t)$

| *Parallel*: $[\![\ i{<}length\ Ts;\ Ts!i = (Some\ c,\ q);\ (Some\ c,\ s)\ {-1\rightarrow}\ (r,\ t)\ ]\!]$
$\Longrightarrow (Parallel\ Ts,\ s)\ {-P1\rightarrow}\ (Parallel\ (Ts\ [i{:=}(r,\ q)]),\ t)$

| *Basic*: $(Basic\ f,\ s)\ {-P1\rightarrow}\ (Parallel\ [],\ f\ s)$

| *Seq1*: $All\text{-}None\ Ts \Longrightarrow (Seq\ (Parallel\ Ts)\ c,\ s)\ {-P1\rightarrow}\ (c,\ s)$
| *Seq2*: $(c0,\ s)\ {-P1\rightarrow}\ (c2,\ t) \Longrightarrow (Seq\ c0\ c1,\ s)\ {-P1\rightarrow}\ (Seq\ c2\ c1,\ t)$

| *CondT*: $s \in b \Longrightarrow (Cond\ b\ c1\ c2,\ s)\ {-P1\rightarrow}\ (c1,\ s)$
| *CondF*: $s \notin b \Longrightarrow (Cond\ b\ c1\ c2,\ s)\ {-P1\rightarrow}\ (c2,\ s)$

| *WhileF*: $s \notin b \Longrightarrow (While\ b\ i\ c,\ s)\ {-P1\rightarrow}\ (Parallel\ [],\ s)$
| *WhileT*: $s \in b \Longrightarrow (While\ b\ i\ c,\ s)\ {-P1\rightarrow}\ (Seq\ c\ (While\ b\ i\ c),\ s)$

monos *rtrancl-mono*

The corresponding abbreviations are:

abbreviation
$ann\text{-}transition\text{-}n :: ('a\ ann\text{-}com\text{-}op \times 'a) \Rightarrow nat \Rightarrow ('a\ ann\text{-}com\text{-}op \times 'a)$
$\Rightarrow bool\ \ (\text{-}\ {-\text{-}\rightarrow}\ \text{-}[81,81]\ 100)$ **where**
$con\text{-}0\ {-n\rightarrow}\ con\text{-}1 \equiv (con\text{-}0,\ con\text{-}1) \in ann\text{-}transition\ \hat{}\hat{}\ n$

abbreviation
$ann\text{-}transitions :: ('a\ ann\text{-}com\text{-}op \times 'a) \Rightarrow ('a\ ann\text{-}com\text{-}op \times 'a) \Rightarrow bool$
$(\text{-}\ {-{*}\rightarrow}\ \text{-}[81,81]\ 100)$ **where**
$con\text{-}0\ {-{*}\rightarrow}\ con\text{-}1 \equiv (con\text{-}0,\ con\text{-}1) \in ann\text{-}transition^*$

abbreviation
$transition\text{-}n :: ('a\ com \times 'a) \Rightarrow nat \Rightarrow ('a\ com \times 'a) \Rightarrow bool$
$(\text{-}\ {-P\text{-}\rightarrow}\ \text{-}[81,81,81]\ 100)$ **where**
$con\text{-}0\ {-Pn\rightarrow}\ con\text{-}1 \equiv (con\text{-}0,\ con\text{-}1) \in transition\ \hat{}\hat{}\ n$

1.2.2 Definition of Semantics

definition $ann\text{-}sem :: 'a\ ann\text{-}com \Rightarrow 'a \Rightarrow 'a\ set$ **where**
$ann\text{-}sem\ c \equiv \lambda s.\ \{t.\ (Some\ c,\ s)\ {-{*}\rightarrow}\ (None,\ t)\}$

definition $\mathit{ann\text{-}SEM} :: {'a}\ \mathit{ann\text{-}com} \Rightarrow {'a}\ \mathit{set} \Rightarrow {'a}\ \mathit{set}$ **where**
$\mathit{ann\text{-}SEM}\ c\ S \equiv \bigcup(\mathit{ann\text{-}sem}\ c\ \text{`}\ S)$

definition $\mathit{sem} :: {'a}\ \mathit{com} \Rightarrow {'a} \Rightarrow {'a}\ \mathit{set}$ **where**
$\mathit{sem}\ c \equiv \lambda s.\ \{t.\ \exists\, \mathit{Ts}.\ (c,\ s)\ {-P{*}{\rightarrow}}\ (\mathit{Parallel}\ \mathit{Ts},\ t) \wedge \mathit{All\text{-}None}\ \mathit{Ts}\}$

definition $\mathit{SEM} :: {'a}\ \mathit{com} \Rightarrow {'a}\ \mathit{set} \Rightarrow {'a}\ \mathit{set}$ **where**
$\mathit{SEM}\ c\ S \equiv \bigcup(\mathit{sem}\ c\ \text{`}\ S)$

abbreviation $\mathit{Omega} :: {'a}\ \mathit{com} \quad (\Omega\ \mathit{63})$
where $\Omega \equiv \mathit{While}\ \mathit{UNIV}\ \mathit{UNIV}\ (\mathit{Basic}\ \mathit{id})$

primrec $\mathit{fwhile} :: {'a}\ \mathit{bexp} \Rightarrow {'a}\ \mathit{com} \Rightarrow \mathit{nat} \Rightarrow {'a}\ \mathit{com}$ **where**
$\mathit{fwhile}\ b\ c\ \mathit{0} = \Omega$
$|\ \mathit{fwhile}\ b\ c\ (\mathit{Suc}\ n) = \mathit{Cond}\ b\ (\mathit{Seq}\ c\ (\mathit{fwhile}\ b\ c\ n))\ (\mathit{Basic}\ \mathit{id})$

Proofs

declare *ann-transition-transition.intros* [*intro*]
inductive-cases *transition-cases*:
$(\mathit{Parallel}\ T,s)\ {-P1{\rightarrow}}\ t$
$(\mathit{Basic}\ f,\ s)\ {-P1{\rightarrow}}\ t$
$(\mathit{Seq}\ c1\ c2,\ s)\ {-P1{\rightarrow}}\ t$
$(\mathit{Cond}\ b\ c1\ c2,\ s)\ {-P1{\rightarrow}}\ t$
$(\mathit{While}\ b\ i\ c,\ s)\ {-P1{\rightarrow}}\ t$

lemma *Parallel-empty-lemma* [*rule-format* (*no-asm*)]:
$(\mathit{Parallel}\ [],s)\ {-Pn{\rightarrow}}\ (\mathit{Parallel}\ \mathit{Ts},t) \longrightarrow \mathit{Ts}=[] \wedge n=0 \wedge s=t$
⟨*proof*⟩

lemma *Parallel-AllNone-lemma* [*rule-format* (*no-asm*)]:
$\mathit{All\text{-}None}\ \mathit{Ss} \longrightarrow (\mathit{Parallel}\ \mathit{Ss},s)\ {-Pn{\rightarrow}}\ (\mathit{Parallel}\ \mathit{Ts},t) \longrightarrow \mathit{Ts}=\mathit{Ss} \wedge n=0 \wedge s=t$
⟨*proof*⟩

lemma *Parallel-AllNone*: $\mathit{All\text{-}None}\ \mathit{Ts} \Longrightarrow (\mathit{SEM}\ (\mathit{Parallel}\ \mathit{Ts})\ X) = X$
⟨*proof*⟩

lemma *Parallel-empty*: $\mathit{Ts}=[] \Longrightarrow (\mathit{SEM}\ (\mathit{Parallel}\ \mathit{Ts})\ X) = X$
⟨*proof*⟩

Set of lemmas from Apt and Olderog "Verification of sequential and concurrent programs", page 63.

lemma *L3-5i*: $X \subseteq Y \Longrightarrow \mathit{SEM}\ c\ X \subseteq \mathit{SEM}\ c\ Y$
⟨*proof*⟩

lemma *L3-5ii-lemma1*:
$[\![\ (c1,\ s1)\ {-P{*}{\rightarrow}}\ (\mathit{Parallel}\ \mathit{Ts},\ s2);\ \mathit{All\text{-}None}\ \mathit{Ts};$
$(c2,\ s2)\ {-P{*}{\rightarrow}}\ (\mathit{Parallel}\ \mathit{Ss},\ s3);\ \mathit{All\text{-}None}\ \mathit{Ss}\ ]\!]$
$\Longrightarrow (\mathit{Seq}\ c1\ c2,\ s1)\ {-P{*}{\rightarrow}}\ (\mathit{Parallel}\ \mathit{Ss},\ s3)$

⟨*proof*⟩

lemma *L3-5ii-lemma2* [*rule-format* (*no-asm*)]:
$\forall$ *c1 c2 s t*. (*Seq c1 c2*, *s*) $-Pn\rightarrow$ (*Parallel Ts*, *t*) $\longrightarrow$
(*All-None Ts*) $\longrightarrow$ ($\exists$ *y m Rs*. (*c1*,*s*) $-P{*}\rightarrow$ (*Parallel Rs*, *y*) $\wedge$
(*All-None Rs*) $\wedge$ (*c2*, *y*) $-Pm\rightarrow$ (*Parallel Ts*, *t*) $\wedge$ $m \leq n$)
⟨*proof*⟩

lemma *L3-5ii-lemma3*:
⟦(*Seq c1 c2*,*s*) $-P{*}\rightarrow$ (*Parallel Ts*,*t*); *All-None Ts*⟧ $\Longrightarrow$
($\exists$ *y Rs*. (*c1*,*s*) $-P{*}\rightarrow$ (*Parallel Rs*,*y*) $\wedge$ *All-None Rs*
$\wedge$ (*c2*,*y*) $-P{*}\rightarrow$ (*Parallel Ts*,*t*))
⟨*proof*⟩

lemma *L3-5ii*: *SEM* (*Seq c1 c2*) *X* = *SEM c2* (*SEM c1 X*)
⟨*proof*⟩

lemma *L3-5iii*: *SEM* (*Seq* (*Seq c1 c2*) *c3*) *X* = *SEM* (*Seq c1* (*Seq c2 c3*)) *X*
⟨*proof*⟩

lemma *L3-5iv*:
SEM (*Cond b c1 c2*) *X* = (*SEM c1* ($X \cap b$)) *Un* (*SEM c2* ($X \cap (-b)$))
⟨*proof*⟩

lemma *L3-5v-lemma1*[*rule-format*]:
(*S*,*s*) $-Pn\rightarrow$ (*T*,*t*) $\longrightarrow$ $S{=}\Omega$ $\longrightarrow$ ($\neg$($\exists$ *Rs*. *T*=(*Parallel Rs*) $\wedge$ *All-None Rs*))
⟨*proof*⟩

lemma *L3-5v-lemma2*: ⟦(Ω, *s*) $-P{*}\rightarrow$ (*Parallel Ts*, *t*); *All-None Ts* ⟧ $\Longrightarrow$ *False*
⟨*proof*⟩

lemma *L3-5v-lemma3*: *SEM* (Ω) *S* = {}
⟨*proof*⟩

lemma *L3-5v-lemma4* [*rule-format*]:
$\forall$ *s*. (*While b i c*, *s*) $-Pn\rightarrow$ (*Parallel Ts*, *t*) $\longrightarrow$ *All-None Ts* $\longrightarrow$
($\exists$ *k*. (*fwhile b c k*, *s*) $-P{*}\rightarrow$ (*Parallel Ts*, *t*))
⟨*proof*⟩

lemma *L3-5v-lemma5* [*rule-format*]:
$\forall$ *s*. (*fwhile b c k*, *s*) $-P{*}\rightarrow$ (*Parallel Ts*, *t*) $\longrightarrow$ *All-None Ts* $\longrightarrow$
(*While b i c*, *s*) $-P{*}\rightarrow$ (*Parallel Ts*,*t*)
⟨*proof*⟩

lemma *L3-5v*: *SEM* (*While b i c*) = (λx. ($\bigcup k$. *SEM* (*fwhile b c k*) *x*))
⟨*proof*⟩

1.3 Validity of Correctness Formulas

definition *com-validity* :: $'a$ *assn* $\Rightarrow$ $'a$ *com* $\Rightarrow$ $'a$ *assn* $\Rightarrow$ *bool* ((*3*$\|$= -// -//-) [*90*,*55*,*90*] *50*) **where**
$\|= p\ c\ q \equiv SEM\ c\ p \subseteq q$

definition *ann-com-validity* :: $'a$ *ann-com* $\Rightarrow$ $'a$ *assn* $\Rightarrow$ *bool* ($\models$ - - [*60*,*90*] *45*) **where**
$\models c\ q \equiv ann\text{-}SEM\ c\ (pre\ c) \subseteq q$

end

1.4 The Proof System

theory *OG-Hoare* **imports** *OG-Tran* **begin**

primrec *assertions* :: $'a$ *ann-com* $\Rightarrow$ ($'a$ *assn*) *set* **where**
assertions (*AnnBasic r f*) = $\{r\}$
| *assertions* (*AnnSeq c1 c2*) = *assertions c1* $\cup$ *assertions c2*
| *assertions* (*AnnCond1 r b c1 c2*) = $\{r\}$ $\cup$ *assertions c1* $\cup$ *assertions c2*
| *assertions* (*AnnCond2 r b c*) = $\{r\}$ $\cup$ *assertions c*
| *assertions* (*AnnWhile r b i c*) = $\{r, i\}$ $\cup$ *assertions c*
| *assertions* (*AnnAwait r b c*) = $\{r\}$

primrec *atomics* :: $'a$ *ann-com* $\Rightarrow$ ($'a$ *assn* $\times$ $'a$ *com*) *set* **where**
atomics (*AnnBasic r f*) = $\{(r,\ Basic\ f)\}$
| *atomics* (*AnnSeq c1 c2*) = *atomics c1* $\cup$ *atomics c2*
| *atomics* (*AnnCond1 r b c1 c2*) = *atomics c1* $\cup$ *atomics c2*
| *atomics* (*AnnCond2 r b c*) = *atomics c*
| *atomics* (*AnnWhile r b i c*) = *atomics c*
| *atomics* (*AnnAwait r b c*) = $\{(r \cap b,\ c)\}$

primrec *com* :: $'a$ *ann-triple-op* $\Rightarrow$ $'a$ *ann-com-op* **where**
com (*c*, *q*) = *c*

primrec *post* :: $'a$ *ann-triple-op* $\Rightarrow$ $'a$ *assn* **where**
post (*c*, *q*) = *q*

definition *interfree-aux* :: ($'a$ *ann-com-op* $\times$ $'a$ *assn* $\times$ $'a$ *ann-com-op*) $\Rightarrow$ *bool* **where**
interfree-aux $\equiv$ λ(*co*, *q*, *co'*). *co'*= *None* $\vee$
($\forall$ (*r*,*a*) $\in$ *atomics* (*the co'*). $\|$= ($q \cap r$) *a q* $\wedge$
(*co* = *None* $\vee$ ($\forall$ *p* $\in$ *assertions* (*the co*). $\|$= ($p \cap r$) *a p*)))

definition *interfree* :: ((($'a$ *ann-triple-op*) *list*) $\Rightarrow$ *bool* **where**
interfree Ts $\equiv$ $\forall$ *i j*. *i* < *length Ts* $\wedge$ *j* < *length Ts* $\wedge$ $i \neq j$ $\longrightarrow$
interfree-aux (*com* (*Ts*!*i*), *post* (*Ts*!*i*), *com* (*Ts*!*j*))

inductive

$oghoare :: 'a\ assn \Rightarrow 'a\ com \Rightarrow 'a\ assn \Rightarrow bool\ \ ((3\|- -//-//-)\ [90,55,90]\ 50)$
and $ann\text{-}hoare :: 'a\ ann\text{-}com \Rightarrow 'a\ assn \Rightarrow bool\ \ ((2\vdash -//\ -)\ [60,90]\ 45)$
where
$AnnBasic$: $r \subseteq \{s.\ f\ s \in q\} \Longrightarrow \vdash (AnnBasic\ r\ f)\ q$

| $AnnSeq$: $[\![\vdash c0\ pre\ c1; \vdash c1\ q\ ]\!] \Longrightarrow \vdash (AnnSeq\ c0\ c1)\ q$

| $AnnCond1$: $[\![\ r \cap b \subseteq pre\ c1; \vdash c1\ q;\ r \cap -b \subseteq pre\ c2; \vdash c2\ q]\!]$
$\Longrightarrow \vdash (AnnCond1\ r\ b\ c1\ c2)\ q$
| $AnnCond2$: $[\![\ r \cap b \subseteq pre\ c; \vdash c\ q;\ r \cap -b \subseteq q\ ]\!] \Longrightarrow \vdash (AnnCond2\ r\ b\ c)\ q$

| $AnnWhile$: $[\![\ r \subseteq i;\ i \cap b \subseteq pre\ c; \vdash c\ i;\ i \cap -b \subseteq q\ ]\!]$
$\Longrightarrow \vdash (AnnWhile\ r\ b\ i\ c)\ q$

| $AnnAwait$: $[\![\ atom\text{-}com\ c;\ \|- (r \cap b)\ c\ q\ ]\!] \Longrightarrow \vdash (AnnAwait\ r\ b\ c)\ q$

| $AnnConseq$: $[\![\vdash c\ q;\ q \subseteq q'\ ]\!] \Longrightarrow \vdash c\ q'$

| $Parallel$: $[\![\ \forall i{<}length\ Ts.\ \exists c\ q.\ Ts!i = (Some\ c,\ q) \wedge \vdash c\ q;\ interfree\ Ts\ ]\!]$
$\Longrightarrow \|- (\bigcap i{\in}\{i.\ i{<}length\ Ts\}.\ pre(the(com(Ts!i))))$
$Parallel\ Ts$
$(\bigcap i{\in}\{i.\ i{<}length\ Ts\}.\ post(Ts!i))$

| $Basic$: $\|- \{s.\ f\ s \in q\}\ (Basic\ f)\ q$

| Seq: $[\![\ \|- p\ c1\ r;\ \|- r\ c2\ q\ ]\!] \Longrightarrow \|- p\ (Seq\ c1\ c2)\ q$

| $Cond$: $[\![\ \|- (p \cap b)\ c1\ q;\ \|- (p \cap -b)\ c2\ q\ ]\!] \Longrightarrow \|- p\ (Cond\ b\ c1\ c2)\ q$

| $While$: $[\![\ \|- (p \cap b)\ c\ p\ ]\!] \Longrightarrow \|- p\ (While\ b\ i\ c)\ (p \cap -b)$

| $Conseq$: $[\![\ p' \subseteq p;\ \|- p\ c\ q\ ;\ q \subseteq q'\ ]\!] \Longrightarrow \|- p'\ c\ q'$

1.5 Soundness

lemmas *[cong del] = if-weak-cong*

lemmas *ann-hoare-induct = oghoare-ann-hoare.induct [THEN conjunct2]*
lemmas *oghoare-induct = oghoare-ann-hoare.induct [THEN conjunct1]*

lemmas *AnnBasic = oghoare-ann-hoare.AnnBasic*
lemmas *AnnSeq = oghoare-ann-hoare.AnnSeq*
lemmas *AnnCond1 = oghoare-ann-hoare.AnnCond1*
lemmas *AnnCond2 = oghoare-ann-hoare.AnnCond2*
lemmas *AnnWhile = oghoare-ann-hoare.AnnWhile*
lemmas *AnnAwait = oghoare-ann-hoare.AnnAwait*
lemmas *AnnConseq = oghoare-ann-hoare.AnnConseq*

lemmas *Parallel* = *oghoare-ann-hoare.Parallel*
lemmas *Basic* = *oghoare-ann-hoare.Basic*
lemmas *Seq* = *oghoare-ann-hoare.Seq*
lemmas *Cond* = *oghoare-ann-hoare.Cond*
lemmas *While* = *oghoare-ann-hoare.While*
lemmas *Conseq* = *oghoare-ann-hoare.Conseq*

1.5.1 Soundness of the System for Atomic Programs

lemma *Basic-ntran* [*rule-format*]:
$(Basic\ f,\ s)\ -Pn\rightarrow\ (Parallel\ Ts,\ t)\ \longrightarrow\ All\text{-}None\ Ts\ \longrightarrow\ t = f\ s$
⟨*proof*⟩

lemma *SEM-fwhile*: $SEM\ S\ (p \cap b) \subseteq p \Longrightarrow SEM\ (fwhile\ b\ S\ k)\ p \subseteq (p \cap -b)$
⟨*proof*⟩

lemma *atom-hoare-sound* [*rule-format*]:
$\|-\ p\ c\ q \longrightarrow atom\text{-}com(c) \longrightarrow \|=\ p\ c\ q$
⟨*proof*⟩

1.5.2 Soundness of the System for Component Programs

inductive-cases *ann-transition-cases*:
$(None,s)\ -1\rightarrow\ (c',\ s')$
$(Some\ (AnnBasic\ r\ f),s)\ -1\rightarrow\ (c',\ s')$
$(Some\ (AnnSeq\ c1\ c2),\ s)\ -1\rightarrow\ (c',\ s')$
$(Some\ (AnnCond1\ r\ b\ c1\ c2),\ s)\ -1\rightarrow\ (c',\ s')$
$(Some\ (AnnCond2\ r\ b\ c),\ s)\ -1\rightarrow\ (c',\ s')$
$(Some\ (AnnWhile\ r\ b\ I\ c),\ s)\ -1\rightarrow\ (c',\ s')$
$(Some\ (AnnAwait\ r\ b\ c),s)\ -1\rightarrow\ (c',\ s')$

Strong Soundness for Component Programs:

lemma *ann-hoare-case-analysis* [*rule-format*]: $\vdash\ C\ q' \longrightarrow$
$((\forall\, r\, f.\ C = AnnBasic\ r\ f \longrightarrow (\exists\, q.\ r \subseteq \{s.\ f\ s \in q\} \wedge q \subseteq q')) \wedge$
$(\forall\, c0\ c1.\ C = AnnSeq\ c0\ c1 \longrightarrow (\exists\, q.\ q \subseteq q' \wedge \vdash c0\ pre\ c1 \wedge \vdash c1\ q)) \wedge$
$(\forall\, r\ b\ c1\ c2.\ C = AnnCond1\ r\ b\ c1\ c2 \longrightarrow (\exists\, q.\ q \subseteq q' \wedge$
$r \cap b \subseteq pre\ c1 \wedge \vdash c1\ q \wedge r \cap -b \subseteq pre\ c2 \wedge \vdash c2\ q)) \wedge$
$(\forall\, r\ b\ c.\ C = AnnCond2\ r\ b\ c \longrightarrow$
$(\exists\, q.\ q \subseteq q' \wedge r \cap b \subseteq pre\ c\ \wedge \vdash c\ q \wedge r \cap -b \subseteq q)) \wedge$
$(\forall\, r\ i\ b\ c.\ C = AnnWhile\ r\ b\ i\ c \longrightarrow$
$(\exists\, q.\ q \subseteq q' \wedge r \subseteq i \wedge i \cap b \subseteq pre\ c \wedge \vdash c\ i \wedge i \cap -b \subseteq q)) \wedge$
$(\forall\, r\ b\ c.\ C = AnnAwait\ r\ b\ c \longrightarrow (\exists\, q.\ q \subseteq q' \wedge \|-\ (r \cap b)\ c\ q)))$
⟨*proof*⟩

lemma *Help*: $(transition \cap \{(x,y).\ True\}) = (transition)$
⟨*proof*⟩

lemma *Strong-Soundness-aux-aux* [*rule-format*]:
$(co,\ s)\ -1\rightarrow\ (co',\ t) \longrightarrow (\forall\, c.\ co = Some\ c \longrightarrow s \in pre\ c \longrightarrow$

$(\forall q. \vdash c\ q \longrightarrow (\mathit{if}\ co' = \mathit{None}\ \mathit{then}\ t{\in}q\ \mathit{else}\ t \in \mathit{pre}(\mathit{the}\ co') \wedge \vdash (\mathit{the}\ co')\ q\ )))$
⟨*proof*⟩

lemma *Strong-Soundness-aux*: $[\![\ (\mathit{Some}\ c,\ s)\ {-}{*}{\rightarrow}\ (co,\ t);\ s \in \mathit{pre}\ c; \vdash c\ q\ ]\!]$
$\Longrightarrow \mathit{if}\ co = \mathit{None}\ \mathit{then}\ t \in q\ \mathit{else}\ t \in \mathit{pre}\ (\mathit{the}\ co) \wedge \vdash (\mathit{the}\ co)\ q$
⟨*proof*⟩

lemma *Strong-Soundness*: $[\![\ (\mathit{Some}\ c,\ s){-}{*}{\rightarrow}(co,\ t);\ s \in \mathit{pre}\ c; \vdash c\ q\ ]\!]$
$\Longrightarrow \mathit{if}\ co = \mathit{None}\ \mathit{then}\ t{\in}q\ \mathit{else}\ t \in \mathit{pre}\ (\mathit{the}\ co)$
⟨*proof*⟩

lemma *ann-hoare-sound*: $\vdash c\ q\ \Longrightarrow\ \models c\ q$
⟨*proof*⟩

1.5.3 Soundness of the System for Parallel Programs

lemma *Parallel-length-post-P1*: $(\mathit{Parallel}\ Ts,s)\ {-}P1{\rightarrow}\ (R',\ t) \Longrightarrow$
$(\exists Rs.\ R' = (\mathit{Parallel}\ Rs) \wedge (\mathit{length}\ Rs) = (\mathit{length}\ Ts) \wedge$
$(\forall i.\ i{<}\mathit{length}\ Ts \longrightarrow \mathit{post}(Rs\ !\ i) = \mathit{post}(Ts\ !\ i)))$
⟨*proof*⟩

lemma *Parallel-length-post-PStar*: $(\mathit{Parallel}\ Ts,s)\ {-}P{*}{\rightarrow}\ (R',t) \Longrightarrow$
$(\exists Rs.\ R' = (\mathit{Parallel}\ Rs) \wedge (\mathit{length}\ Rs) = (\mathit{length}\ Ts) \wedge$
$(\forall i.\ i{<}\mathit{length}\ Ts \longrightarrow \mathit{post}(Ts\ !\ i) = \mathit{post}(Rs\ !\ i)))$
⟨*proof*⟩

lemma *assertions-lemma*: $\mathit{pre}\ c \in \mathit{assertions}\ c$
⟨*proof*⟩

lemma *interfree-aux1* [*rule-format*]:
$(c,s)\ {-}1{\rightarrow}\ (r,t)\ \longrightarrow (\mathit{interfree\text{-}aux}(c1,\ q1,\ c) \longrightarrow \mathit{interfree\text{-}aux}(c1,\ q1,\ r))$
⟨*proof*⟩

lemma *interfree-aux2* [*rule-format*]:
$(c,s)\ {-}1{\rightarrow}\ (r,t) \longrightarrow (\mathit{interfree\text{-}aux}(c,\ q,\ a)\ \longrightarrow \mathit{interfree\text{-}aux}(r,\ q,\ a)\ )$
⟨*proof*⟩

lemma *interfree-lemma*: $[\![\ (\mathit{Some}\ c,\ s)\ {-}1{\rightarrow}\ (r,\ t);\mathit{interfree}\ Ts\ ;\ i{<}\mathit{length}\ Ts;$
$Ts!i = (\mathit{Some}\ c,\ q)\ ]\!] \Longrightarrow \mathit{interfree}\ (Ts[i{:=}\ (r,\ q)])$
⟨*proof*⟩

Strong Soundness Theorem for Parallel Programs:

lemma *Parallel-Strong-Soundness-Seq-aux*:
$[\![\mathit{interfree}\ Ts;\ i{<}\mathit{length}\ Ts;\ \mathit{com}(Ts\ !\ i) = \mathit{Some}(\mathit{AnnSeq}\ c0\ c1)\ ]\!]$
$\Longrightarrow\ \ \mathit{interfree}\ (Ts[i{:=}(\mathit{Some}\ c0,\ \mathit{pre}\ c1)])$
⟨*proof*⟩

lemma *Parallel-Strong-Soundness-Seq* [*rule-format* (*no-asm*)]:
$[\![\ \forall i{<}\mathit{length}\ Ts.\ (\mathit{if}\ \mathit{com}(Ts!i) = \mathit{None}\ \mathit{then}\ b \in \mathit{post}(Ts!i)$

else b ∈ pre(the(com(Ts!i))) ∧ ⊢ the(com(Ts!i)) post(Ts!i));
com(Ts ! i) = Some(AnnSeq c0 c1); i<length Ts; interfree Ts ⟧ ⟹
(∀ *ia<length Ts. (if com(Ts[i:=(Some c0, pre c1)]! ia) = None*
then b ∈ post(Ts[i:=(Some c0, pre c1)]! ia)
else b ∈ pre(the(com(Ts[i:=(Some c0, pre c1)]! ia))) ∧
⊢ the(com(Ts[i:=(Some c0, pre c1)]! ia)) post(Ts[i:=(Some c0, pre c1)]! ia)))
∧ interfree (Ts[i:= (Some c0, pre c1)])
⟨*proof*⟩

lemma *Parallel-Strong-Soundness-aux-aux* [*rule-format*]:
(Some c, b) −1→ (co, t) ⟶
(∀ Ts. i<length Ts ⟶ com(Ts ! i) = Some c ⟶
(∀ i<length Ts. (if com(Ts ! i) = None then b∈post(Ts!i)
else b∈pre(the(com(Ts!i))) ∧ ⊢ the(com(Ts!i)) post(Ts!i))) ⟶
interfree Ts ⟶
(∀ j. j<length Ts ∧ i≠j ⟶ (if com(Ts!j) = None then t∈post(Ts!j)
else t∈pre(the(com(Ts!j))) ∧ ⊢ the(com(Ts!j)) post(Ts!j))))
⟨*proof*⟩

lemma *Parallel-Strong-Soundness-aux* [*rule-format*]:
⟦*(Ts',s) −P∗→ (Rs',t); Ts' = (Parallel Ts); interfree Ts;*
∀ i. i<length Ts ⟶ (∃ c q. (Ts ! i) = (Some c, q) ∧ s∈(pre c) ∧ ⊢ c q) ⟧ ⟹
∀ Rs. Rs' = (Parallel Rs) ⟶ (∀ j. j<length Rs ⟶
(if com(Rs ! j) = None then t∈post(Ts ! j)
else t∈pre(the(com(Rs ! j))) ∧ ⊢ the(com(Rs ! j)) post(Ts ! j))) ∧ interfree Rs
⟨*proof*⟩

lemma *Parallel-Strong-Soundness*:
⟦*(Parallel Ts, s) −P∗→ (Parallel Rs, t); interfree Ts; j<length Rs;*
∀ i. i<length Ts ⟶ (∃ c q. Ts ! i = (Some c, q) ∧ s∈pre c ∧ ⊢ c q) ⟧ ⟹
if com(Rs ! j) = None then t∈post(Ts ! j) else t∈pre (the(com(Rs ! j)))
⟨*proof*⟩

lemma *oghoare-sound* [*rule-format*]: *‖− p c q ⟶ ‖= p c q*
⟨*proof*⟩

end

1.6 Generation of Verification Conditions

theory *OG-Tactics*
imports *OG-Hoare*
begin

lemmas *ann-hoare-intros=AnnBasic AnnSeq AnnCond1 AnnCond2 AnnWhile AnnAwait AnnConseq*
lemmas *oghoare-intros=Parallel Basic Seq Cond While Conseq*

lemma *ParallelConseqRule*:

$[\![\ p \subseteq (\bigcap i{\in}\{i.\ i{<}length\ Ts\}.\ pre(the(com(Ts\ !\ i))));$
$\parallel\!-\ (\bigcap i{\in}\{i.\ i{<}length\ Ts\}.\ pre(the(com(Ts\ !\ i))))$
$(Parallel\ Ts)$
$(\bigcap i{\in}\{i.\ i{<}length\ Ts\}.\ post(Ts\ !\ i));$
$(\bigcap i{\in}\{i.\ i{<}length\ Ts\}.\ post(Ts\ !\ i)) \subseteq q\ ]\!]$
$\Longrightarrow \parallel\!-\ p\ (Parallel\ Ts)\ q$
$\langle proof \rangle$

lemma *SkipRule*: $p \subseteq q \Longrightarrow \parallel\!-\ p\ (Basic\ id)\ q$
$\langle proof \rangle$

lemma *BasicRule*: $p \subseteq \{s.\ (f\ s){\in}q\} \Longrightarrow \parallel\!-\ p\ (Basic\ f)\ q$
$\langle proof \rangle$

lemma *SeqRule*: $[\![\ \parallel\!-\ p\ c1\ r;\ \parallel\!-\ r\ c2\ q\ ]\!] \Longrightarrow \parallel\!-\ p\ (Seq\ c1\ c2)\ q$
$\langle proof \rangle$

lemma *CondRule*:
$[\![\ p \subseteq \{s.\ (s{\in}b \longrightarrow s{\in}w) \wedge (s{\notin}b \longrightarrow s{\in}w')\};\ \parallel\!-\ w\ c1\ q;\ \parallel\!-\ w'\ c2\ q\ ]\!]$
$\Longrightarrow \parallel\!-\ p\ (Cond\ b\ c1\ c2)\ q$
$\langle proof \rangle$

lemma *WhileRule*: $[\![\ p \subseteq i;\ \parallel\!-\ (i \cap b)\ c\ i\ ;\ (i \cap (-b)) \subseteq q\ ]\!]$
$\Longrightarrow \parallel\!-\ p\ (While\ b\ i\ c)\ q$
$\langle proof \rangle$

Three new proof rules for special instances of the *AnnBasic* and the *AnnAwait* commands when the transformation performed on the state is the identity, and for an *AnnAwait* command where the boolean condition is $\{s.\ True\}$:

lemma *AnnatomRule*:
$[\![\ atom\text{-}com(c);\ \parallel\!-\ r\ c\ q\ ]\!] \Longrightarrow \vdash (AnnAwait\ r\ \{s.\ True\}\ c)\ q$
$\langle proof \rangle$

lemma *AnnskipRule*:
$r \subseteq q \Longrightarrow \vdash (AnnBasic\ r\ id)\ q$
$\langle proof \rangle$

lemma *AnnwaitRule*:
$[\![\ (r \cap b) \subseteq q\ ]\!] \Longrightarrow \vdash (AnnAwait\ r\ b\ (Basic\ id))\ q$
$\langle proof \rangle$

Lemmata to avoid using the definition of *map-ann-hoare*, *interfree-aux*, *interfree-swap* and *interfree* by splitting it into different cases:

lemma *interfree-aux-rule1*: $interfree\text{-}aux(co,\ q,\ None)$
$\langle proof \rangle$

lemma *interfree-aux-rule2*:
$\forall (R,r){\in}(atomics\ a).\ \parallel\!-\ (q \cap R)\ r\ q \Longrightarrow interfree\text{-}aux(None,\ q,\ Some\ a)$

$\langle proof \rangle$

lemma *interfree-aux-rule3*:
$(\forall (R, r)\in(atomics\ a).\ \|- (q \cap R)\ r\ q \wedge (\forall p\in(assertions\ c).\ \|- (p \cap R)\ r\ p))$
$\Longrightarrow interfree\text{-}aux(Some\ c,\ q,\ Some\ a)$
$\langle proof \rangle$

lemma *AnnBasic-assertions*:
$[\![interfree\text{-}aux(None,\ r,\ Some\ a);\ interfree\text{-}aux(None,\ q,\ Some\ a)]\!] \Longrightarrow$
$interfree\text{-}aux(Some\ (AnnBasic\ r\ f),\ q,\ Some\ a)$
$\langle proof \rangle$

lemma *AnnSeq-assertions*:
$[\![\ interfree\text{-}aux(Some\ c1,\ q,\ Some\ a);\ interfree\text{-}aux(Some\ c2,\ q,\ Some\ a)]\!]\Longrightarrow$
$interfree\text{-}aux(Some\ (AnnSeq\ c1\ c2),\ q,\ Some\ a)$
$\langle proof \rangle$

lemma *AnnCond1-assertions*:
$[\![\ interfree\text{-}aux(None,\ r,\ Some\ a);\ interfree\text{-}aux(Some\ c1,\ q,\ Some\ a);$
$interfree\text{-}aux(Some\ c2,\ q,\ Some\ a)]\!]\Longrightarrow$
$interfree\text{-}aux(Some(AnnCond1\ r\ b\ c1\ c2),\ q,\ Some\ a)$
$\langle proof \rangle$

lemma *AnnCond2-assertions*:
$[\![\ interfree\text{-}aux(None,\ r,\ Some\ a);\ interfree\text{-}aux(Some\ c,\ q,\ Some\ a)]\!]\Longrightarrow$
$interfree\text{-}aux(Some\ (AnnCond2\ r\ b\ c),\ q,\ Some\ a)$
$\langle proof \rangle$

lemma *AnnWhile-assertions*:
$[\![\ interfree\text{-}aux(None,\ r,\ Some\ a);\ interfree\text{-}aux(None,\ i,\ Some\ a);$
$interfree\text{-}aux(Some\ c,\ q,\ Some\ a)]\!]\Longrightarrow$
$interfree\text{-}aux(Some\ (AnnWhile\ r\ b\ i\ c),\ q,\ Some\ a)$
$\langle proof \rangle$

lemma *AnnAwait-assertions*:
$[\![\ interfree\text{-}aux(None,\ r,\ Some\ a);\ interfree\text{-}aux(None,\ q,\ Some\ a)]\!]\Longrightarrow$
$interfree\text{-}aux(Some\ (AnnAwait\ r\ b\ c),\ q,\ Some\ a)$
$\langle proof \rangle$

lemma *AnnBasic-atomics*:
$\|- (q \cap r)\ (Basic\ f)\ q \Longrightarrow interfree\text{-}aux(None,\ q,\ Some\ (AnnBasic\ r\ f))$
$\langle proof \rangle$

lemma *AnnSeq-atomics*:
$[\![\ interfree\text{-}aux(Any,\ q,\ Some\ a1);\ interfree\text{-}aux(Any,\ q,\ Some\ a2)]\!]\Longrightarrow$
$interfree\text{-}aux(Any,\ q,\ Some\ (AnnSeq\ a1\ a2))$
$\langle proof \rangle$

lemma *AnnCond1-atomics*:

$\llbracket$ *interfree-aux*(*Any*, *q*, *Some a1*); *interfree-aux*(*Any*, *q*, *Some a2*)$\rrbracket \Longrightarrow$
interfree-aux(*Any*, *q*, *Some* (*AnnCond1 r b a1 a2*))
⟨*proof*⟩

lemma *AnnCond2-atomics*:
interfree-aux (*Any*, *q*, *Some a*)$\Longrightarrow$ *interfree-aux*(*Any*, *q*, *Some* (*AnnCond2 r b a*))
⟨*proof*⟩

lemma *AnnWhile-atomics*: *interfree-aux* (*Any*, *q*, *Some a*)
$\Longrightarrow$ *interfree-aux*(*Any*, *q*, *Some* (*AnnWhile r b i a*))
⟨*proof*⟩

lemma *Annatom-atomics*:
$\Vert-$ ($q \cap r$) *a q* $\Longrightarrow$ *interfree-aux* (*None*, *q*, *Some* (*AnnAwait r* {*x*. *True*} *a*))
⟨*proof*⟩

lemma *AnnAwait-atomics*:
$\Vert-$ ($q \cap (r \cap b)$) *a q* $\Longrightarrow$ *interfree-aux* (*None*, *q*, *Some* (*AnnAwait r b a*))
⟨*proof*⟩

definition *interfree-swap* :: (*'a ann-triple-op* $*$ (*'a ann-triple-op*) *list*) $\Rightarrow$ *bool*
where
interfree-swap == λ(*x*, *xs*). $\forall$ *y*$\in$*set xs*. *interfree-aux* (*com x*, *post x*, *com y*)
$\wedge$ *interfree-aux*(*com y*, *post y*, *com x*)

lemma *interfree-swap-Empty*: *interfree-swap* (*x*, [])
⟨*proof*⟩

lemma *interfree-swap-List*:
$\llbracket$ *interfree-aux* (*com x*, *post x*, *com y*);
interfree-aux (*com y*, *post y* ,*com x*); *interfree-swap* (*x*, *xs*) $\rrbracket$
$\Longrightarrow$ *interfree-swap* (*x*, *y*#*xs*)
⟨*proof*⟩

lemma *interfree-swap-Map*: $\forall$ *k*. *i*$\leq$*k* $\wedge$ *k*<*j* $\longrightarrow$ *interfree-aux* (*com x*, *post x*, *c k*)
$\wedge$ *interfree-aux* (*c k*, *Q k*, *com x*)
$\Longrightarrow$ *interfree-swap* (*x*, *map* (λ*k*. (*c k*, *Q k*)) [*i*..<*j*])
⟨*proof*⟩

lemma *interfree-Empty*: *interfree* []
⟨*proof*⟩

lemma *interfree-List*:
$\llbracket$ *interfree-swap*(*x*, *xs*); *interfree xs* $\rrbracket$ $\Longrightarrow$ *interfree* (*x*#*xs*)
⟨*proof*⟩

lemma *interfree-Map*:

$(\forall\, i\, j.\ a \le i \wedge i < b \wedge a \le j \wedge j < b \ \wedge i \neq j \longrightarrow \mathit{interfree\text{-}aux}\ (c\ i,\ Q\ i,\ c\ j))$
$\Longrightarrow \mathit{interfree}\ (\mathit{map}\ (\lambda k.\ (c\ k,\ Q\ k))\ [a..<b])$
⟨*proof*⟩

definition $\mathit{map\text{-}ann\text{-}hoare} :: (('a\ \mathit{ann\text{-}com\text{-}op} * 'a\ \mathit{assn})\ \mathit{list}) \Rightarrow \mathit{bool}\ \ ([\vdash]\ \text{-}\ [0]\ 45)$
where
$[\vdash]\ \mathit{Ts} == (\forall\, i < \mathit{length}\ \mathit{Ts}.\ \exists\, c\ q.\ \mathit{Ts}!i = (\mathit{Some}\ c,\ q) \wedge \vdash c\ q)$

lemma *MapAnnEmpty*: $[\vdash]\ []$
⟨*proof*⟩

lemma *MapAnnList*: $[\![\vdash c\ q\ ;\ [\vdash]\ \mathit{xs}\ ]\!] \Longrightarrow [\vdash]\ (\mathit{Some}\ c,q)\#\mathit{xs}$
⟨*proof*⟩

lemma *MapAnnMap*:
$\forall\, k.\ i \le k \wedge k < j \longrightarrow \vdash (c\ k)\ (Q\ k) \Longrightarrow [\vdash]\ \mathit{map}\ (\lambda k.\ (\mathit{Some}\ (c\ k),\ Q\ k))\ [i..<j]$
⟨*proof*⟩

lemma *ParallelRule*:$[\![\ [\vdash]\ \mathit{Ts}\ ;\ \mathit{interfree}\ \mathit{Ts}\ ]\!]$
$\Longrightarrow \Vert\!-\ (\bigcap i \in \{i.\ i < \mathit{length}\ \mathit{Ts}\}.\ \mathit{pre}(\mathit{the}(\mathit{com}(\mathit{Ts}!i))))$
Parallel Ts
$(\bigcap i \in \{i.\ i < \mathit{length}\ \mathit{Ts}\}.\ \mathit{post}(\mathit{Ts}!i))$
⟨*proof*⟩

The following are some useful lemmas and simplification tactics to control which theorems are used to simplify at each moment, so that the original input does not suffer any unexpected transformation.

lemma *Compl-Collect*: $-(\mathit{Collect}\ b) = \{x.\ \neg(b\ x)\}$
⟨*proof*⟩

lemma *list-length*: $\mathit{length}\ [] = 0\ \mathit{length}\ (x\#\mathit{xs}) = \mathit{Suc}(\mathit{length}\ \mathit{xs})$
⟨*proof*⟩
lemma *list-lemmas*: $\mathit{length}\ [] = 0\ \mathit{length}\ (x\#\mathit{xs}) = \mathit{Suc}(\mathit{length}\ \mathit{xs})$
$(x\#\mathit{xs})\ !\ 0 = x\ (x\#\mathit{xs})\ !\ \mathit{Suc}\ n = \mathit{xs}\ !\ n$
⟨*proof*⟩
lemma *le-Suc-eq-insert*: $\{i.\ i < \mathit{Suc}\ n\} = \mathit{insert}\ n\ \{i.\ i < n\}$
⟨*proof*⟩
lemmas *primrecdef-list = pre.simps assertions.simps atomics.simps atom-com.simps*
lemmas *my-simp-list = list-lemmas fst-conv snd-conv*
not-less0 refl le-Suc-eq-insert Suc-not-Zero Zero-not-Suc nat.inject
Collect-mem-eq ball-simps option.simps primrecdef-list
lemmas *ParallelConseq-list = INTER-eq Collect-conj-eq length-map length-upt length-append*

⟨*ML*⟩

The following tactic applies *tac* to each conjunct in a subgoal of the form $A \Longrightarrow a1 \wedge a2 \wedge .. \wedge an$ returning n subgoals, one for each conjunct:

⟨*ML*⟩

Tactic for the generation of the verification conditions

The tactic basically uses two subtactics:

HoareRuleTac is called at the level of parallel programs, it uses the ParallelTac to solve parallel composition of programs. This verification has two parts, namely, (1) all component programs are correct and (2) they are interference free. *HoareRuleTac* is also called at the level of atomic regions, i.e. $\langle\ \rangle$ and *AWAIT b THEN - END*, and at each interference freedom test.

AnnHoareRuleTac is for component programs which are annotated programs and so, there are not unknown assertions (no need to use the parameter precond, see NOTE).

NOTE: precond(::bool) informs if the subgoal has the form $\|-$ *?p c q*, in this case we have precond=False and the generated verification condition would have the form *?p* $\subseteq$ ... which can be solved by *rtac subset-refl*, if True we proceed to simplify it using the simplification tactics above.

⟨*ML*⟩

The final tactic is given the name *oghoare*:

⟨*ML*⟩

Notice that the tactic for parallel programs *oghoare-tac* is initially invoked with the value *true* for the parameter *precond*.

Parts of the tactic can be also individually used to generate the verification conditions for annotated sequential programs and to generate verification conditions out of interference freedom tests:

⟨*ML*⟩

The so defined ML tactics are then "exported" to be used in Isabelle proofs.

⟨*ML*⟩

Tactics useful for dealing with the generated verification conditions:

⟨*ML*⟩

end

1.7 Concrete Syntax

theory *Quote-Antiquote* **imports** *Main* **begin**

syntax

-quote	:: $'b \Rightarrow ('a \Rightarrow 'b)$	$((\ll\text{-}\gg)\ [0]\ 1000)$
-antiquote	:: $('a \Rightarrow 'b) \Rightarrow 'b$	$('\text{-}\ [1000]\ 1000)$

-Assert :: *'a* ⇒ *'a set* ((⦃-⦄) [*0*] *1000*)

translations
⦃*b*⦄ ⇀ *CONST Collect* ≪*b*≫

⟨*ML*⟩

end
theory *OG-Syntax*
imports *OG-Tactics Quote-Antiquote*
begin

Syntax for commands and for assertions and boolean expressions in commands *com* and annotated commands *ann-com*.

abbreviation *Skip* :: *'a com* (*SKIP 63*)
where *SKIP* ≡ *Basic id*

abbreviation *AnnSkip* :: *'a assn* ⇒ *'a ann-com* (*-//SKIP* [*90*] *63*)
where *r SKIP* ≡ *AnnBasic r id*

notation
Seq (-,,/ - [*55*, *56*] *55*) **and**
AnnSeq (-;;/ - [*60*,*61*] *60*)

syntax
-Assign :: *idt* ⇒ *'b* ⇒ *'a com* ((´- :=/ -) [*70*, *65*] *61*)
-AnnAssign :: *'a assn* ⇒ *idt* ⇒ *'b* ⇒ *'a com* ((- ´- :=/ -) [*90*,*70*,*65*] *61*)

translations
´*x* := *a* ⇀ *CONST Basic* ≪´(*-update-name x* (λ-. *a*))≫
r ´*x* := *a* ⇀ *CONST AnnBasic r* ≪´(*-update-name x* (λ-. *a*))≫

syntax
-AnnCond1 :: *'a assn* ⇒ *'a bexp* ⇒ *'a ann-com* ⇒ *'a ann-com* ⇒ *'a ann-com*
(- *//IF* - */THEN* - */ELSE* - */FI* [*90*,*0*,*0*,*0*] *61*)
-AnnCond2 :: *'a assn* ⇒ *'a bexp* ⇒ *'a ann-com* ⇒ *'a ann-com*
(- *//IF* - */THEN* - */FI* [*90*,*0*,*0*] *61*)
-AnnWhile :: *'a assn* ⇒ *'a bexp* ⇒ *'a assn* ⇒ *'a ann-com* ⇒ *'a ann-com*
(- *//WHILE* - */INV* - *//DO* -*//OD* [*90*,*0*,*0*,*0*] *61*)
-AnnAwait :: *'a assn* ⇒ *'a bexp* ⇒ *'a com* ⇒ *'a ann-com*
(- *//AWAIT* - */THEN* /- */END* [*90*,*0*,*0*] *61*)
-AnnAtom :: *'a assn* ⇒ *'a com* ⇒ *'a ann-com* (-//⟨-⟩ [*90*,*0*] *61*)
-AnnWait :: *'a assn* ⇒ *'a bexp* ⇒ *'a ann-com* (-*//WAIT* - *END* [*90*,*0*] *61*)

-Cond :: *'a bexp* ⇒ *'a com* ⇒ *'a com* ⇒ *'a com*
((*0IF* -/ *THEN* -/ *ELSE* -/ *FI*) [*0*, *0*, *0*] *61*)
-Cond2 :: *'a bexp* ⇒ *'a com* ⇒ *'a com* (*IF* - *THEN* - *FI* [*0*,*0*] *56*)
-While-inv :: *'a bexp* ⇒ *'a assn* ⇒ *'a com* ⇒ *'a com*
((*0WHILE* -/ *INV* - *//DO* - */OD*) [*0*, *0*, *0*] *61*)

-While :: *'a bexp* ⇒ *'a com* ⇒ *'a com*
((*0WHILE - //DO - /OD*) [*0*, *0*] *61*)

translations
IF b THEN c1 ELSE c2 FI ⇀ *CONST Cond* {|*b*|} *c1 c2*
IF b THEN c FI ⇌ *IF b THEN c ELSE SKIP FI*
WHILE b INV i DO c OD ⇀ *CONST While* {|*b*|} *i c*
WHILE b DO c OD ⇌ *WHILE b INV CONST undefined DO c OD*

r IF b THEN c1 ELSE c2 FI ⇀ *CONST AnnCond1 r* {|*b*|} *c1 c2*
r IF b THEN c FI ⇀ *CONST AnnCond2 r* {|*b*|} *c*
r WHILE b INV i DO c OD ⇀ *CONST AnnWhile r* {|*b*|} *i c*
r AWAIT b THEN c END ⇀ *CONST AnnAwait r* {|*b*|} *c*
r ⟨*c*⟩ ⇌ *r AWAIT CONST True THEN c END*
r WAIT b END ⇌ *r AWAIT b THEN SKIP END*

nonterminal *prgs*

syntax
-PAR :: *prgs* ⇒ *'a* (*COBEGIN//-//COEND* [*57*] *56*)
-prg :: [*'a*, *'a*] ⇒ *prgs* (*-//-* [*60*, *90*] *57*)
-prgs :: [*'a*, *'a*, *prgs*] ⇒ *prgs* (*-//-//*‖*//-* [*60*,*90*,*57*] *57*)

-prg-scheme :: [*'a*, *'a*, *'a*, *'a*, *'a*] ⇒ *prgs*
(*SCHEME* [- ≤ - < -] *-//* - [*0*,*0*,*0*,*60*, *90*] *57*)

translations
-prg c q ⇌ [(*CONST Some c*, *q*)]
-prgs c q ps ⇌ (*CONST Some c*, *q*) # *ps*
-PAR ps ⇌ *CONST Parallel ps*

-prg-scheme j i k c q ⇌ *CONST map* (λ*i*. (*CONST Some c*, *q*)) [*j*..<*k*]

⟨*ML*⟩

end

1.8 Examples

theory *OG-Examples* **imports** *OG-Syntax* **begin**

1.8.1 Mutual Exclusion

Peterson's Algorithm I

Eike Best. "Semantics of Sequential and Parallel Programs", page 217.

record *Petersons-mutex-1* =
pr1 :: *nat*
pr2 :: *nat*

```
 in1 :: bool
 in2 :: bool
 hold :: nat
```

lemma *Petersons-mutex-1*:

```
  ‖− {|´pr1=0 ∧ ¬´in1 ∧ ´pr2=0 ∧ ¬´in2 |}
  COBEGIN {|´pr1=0 ∧ ¬´in1|}
  WHILE True INV {|´pr1=0 ∧ ¬´in1|}
  DO
  {|´pr1=0 ∧ ¬´in1|} ⟨ ´in1:=True,,´pr1:=1 ⟩;;
  {|´pr1=1 ∧ ´in1|}  ⟨ ´hold:=1,,´pr1:=2 ⟩;;
  {|´pr1=2 ∧ ´in1 ∧ (´hold=1 ∨ ´hold=2 ∧ ´pr2=2)|}
  AWAIT (¬´in2 ∨ ¬(´hold=1)) THEN ´pr1:=3 END;;
  {|´pr1=3 ∧ ´in1 ∧ (´hold=1 ∨ ´hold=2 ∧ ´pr2=2)|}
   ⟨´in1:=False,,´pr1:=0⟩
  OD {|´pr1=0 ∧ ¬´in1|}
  ‖
  {|´pr2=0 ∧ ¬´in2|}
  WHILE True INV {|´pr2=0 ∧ ¬´in2|}
  DO
  {|´pr2=0 ∧ ¬´in2|} ⟨ ´in2:=True,,´pr2:=1 ⟩;;
  {|´pr2=1 ∧ ´in2|} ⟨  ´hold:=2,,´pr2:=2 ⟩;;
  {|´pr2=2 ∧ ´in2 ∧ (´hold=2 ∨ (´hold=1 ∧ ´pr1=2))|}
  AWAIT (¬´in1 ∨ ¬(´hold=2)) THEN ´pr2:=3  END;;
  {|´pr2=3 ∧ ´in2 ∧ (´hold=2 ∨ (´hold=1 ∧ ´pr1=2))|}
    ⟨´in2:=False,,´pr2:=0⟩
  OD {|´pr2=0 ∧ ¬´in2|}
  COEND
  {|´pr1=0 ∧ ¬´in1 ∧ ´pr2=0 ∧ ¬´in2|}
⟨proof⟩
```

Peterson's Algorithm II: A Busy Wait Solution

Apt and Olderog. "Verification of sequential and concurrent Programs", page 282.

record *Busy-wait-mutex* =

```
 flag1 :: bool
 flag2 :: bool
 turn  :: nat
 after1 :: bool
 after2 :: bool
```

lemma *Busy-wait-mutex*:

```
 ‖−  {|True|}
  ´flag1:=False,, ´flag2:=False,,
  COBEGIN {|¬´flag1|}
        WHILE True
        INV {|¬´flag1|}
        DO {|¬´flag1|} ⟨ ´flag1:=True,,´after1:=False ⟩;;
```

$\{\!|\acute{}flag1 \wedge \neg\acute{}after1|\!\}$ $\langle$ $\acute{}turn{:=}1,, \acute{}after1{:=}True$ $\rangle$;;
$\{\!|\acute{}flag1 \wedge \acute{}after1 \wedge (\acute{}turn{=}1 \vee \acute{}turn{=}2)|\!\}$
WHILE $\neg(\acute{}flag2 \longrightarrow \acute{}turn{=}2)$
INV $\{\!|\acute{}flag1 \wedge \acute{}after1 \wedge (\acute{}turn{=}1 \vee \acute{}turn{=}2)|\!\}$
DO $\{\!|\acute{}flag1 \wedge \acute{}after1 \wedge (\acute{}turn{=}1 \vee \acute{}turn{=}2)|\!\}$ *SKIP OD*;;
$\{\!|\acute{}flag1 \wedge \acute{}after1 \wedge (\acute{}flag2 \wedge \acute{}after2 \longrightarrow \acute{}turn{=}2)|\!\}$
$\acute{}flag1{:=}False$
OD
$\{\!|False|\!\}$
$\|$
$\{\!|\neg\acute{}flag2|\!\}$
WHILE True
INV $\{\!|\neg\acute{}flag2|\!\}$
DO $\{\!|\neg\acute{}flag2|\!\}$ $\langle$ $\acute{}flag2{:=}True,, \acute{}after2{:=}False$ $\rangle$;;
$\{\!|\acute{}flag2 \wedge \neg\acute{}after2|\!\}$ $\langle$ $\acute{}turn{:=}2,, \acute{}after2{:=}True$ $\rangle$;;
$\{\!|\acute{}flag2 \wedge \acute{}after2 \wedge (\acute{}turn{=}1 \vee \acute{}turn{=}2)|\!\}$
WHILE $\neg(\acute{}flag1 \longrightarrow \acute{}turn{=}1)$
INV $\{\!|\acute{}flag2 \wedge \acute{}after2 \wedge (\acute{}turn{=}1 \vee \acute{}turn{=}2)|\!\}$
DO $\{\!|\acute{}flag2 \wedge \acute{}after2 \wedge (\acute{}turn{=}1 \vee \acute{}turn{=}2)|\!\}$ *SKIP OD*;;
$\{\!|\acute{}flag2 \wedge \acute{}after2 \wedge (\acute{}flag1 \wedge \acute{}after1 \longrightarrow \acute{}turn{=}1)|\!\}$
$\acute{}flag2{:=}False$
OD
$\{\!|False|\!\}$
COEND
$\{\!|False|\!\}$
$\langle proof \rangle$

Peterson's Algorithm III: A Solution using Semaphores

record *Semaphores-mutex* =
out :: *bool*
who :: *nat*

lemma *Semaphores-mutex*:
$\|- \{\!|i{\neq}j|\!\}$
$\acute{}out{:=}True$,,
COBEGIN $\{\!|i{\neq}j|\!\}$
WHILE True INV $\{\!|i{\neq}j|\!\}$
DO $\{\!|i{\neq}j|\!\}$ *AWAIT* $\acute{}out$ *THEN* $\acute{}out{:=}False,, \acute{}who{:=}i$ *END*;;
$\{\!|\neg\acute{}out \wedge \acute{}who{=}i \wedge i{\neq}j|\!\}$ $\acute{}out{:=}True$ *OD*
$\{\!|False|\!\}$
$\|$
$\{\!|i{\neq}j|\!\}$
WHILE True INV $\{\!|i{\neq}j|\!\}$
DO $\{\!|i{\neq}j|\!\}$ *AWAIT* $\acute{}out$ *THEN* $\acute{}out{:=}False,, \acute{}who{:=}j$ *END*;;
$\{\!|\neg\acute{}out \wedge \acute{}who{=}j \wedge i{\neq}j|\!\}$ $\acute{}out{:=}True$ *OD*
$\{\!|False|\!\}$
COEND
$\{\!|False|\!\}$

⟨proof⟩

Peterson's Algorithm III: Parameterized version:

```
lemma Semaphores-parameterized-mutex:
 0<n ⟹ ||− {|True|}
  ´out:=True ,,
 COBEGIN
  SCHEME [0≤ i< n]
   {|True|}
    WHILE True INV {|True|}
     DO {|True|} AWAIT ´out THEN ´out:=False,, ´who:=i END;;
        {|¬´out ∧ ´who=i|} ´out:=True OD
   {|False|}
 COEND
  {|False|}
⟨proof⟩
```

The Ticket Algorithm

```
record Ticket-mutex =
 num :: nat
 nextv :: nat
 turn :: nat list
 index :: nat

lemma Ticket-mutex:
⟦ 0<n; I=«n=length ´turn ∧ 0<´nextv ∧ (∀k l. k<n ∧ l<n ∧ k≠l
    ⟶ ´turn!k < ´num ∧ (´turn!k =0 ∨ ´turn!k≠´turn!l))» ⟧
   ⟹ ||− {|n=length ´turn|}
   ´index:= 0,,
   WHILE ´index < n INV {|n=length ´turn ∧ (∀i<´index. ´turn!i=0)|}
    DO ´turn:= ´turn[´index:=0],, ´index:=´index +1 OD,,
  ´num:=1 ,, ´nextv:=1 ,,
 COBEGIN
  SCHEME [0≤ i< n]
   {|´I|}
    WHILE True INV {|´I|}
     DO {|´I|} ⟨ ´turn :=´turn[i:=´num],, ´num:=´num+1 ⟩;;
        {|´I|} WAIT ´turn!i=´nextv END;;
        {|´I ∧ ´turn!i=´nextv|} ´nextv:=´nextv+1
     OD
   {|False|}
 COEND
  {|False|}
⟨proof⟩
```

1.8.2 Parallel Zero Search

Synchronized Zero Search. Zero-6

Apt and Olderog. "Verification of sequential and concurrent Programs" page 294:

```
record Zero-search =
  turn :: nat
  found :: bool
  x :: nat
  y :: nat

lemma Zero-search:
 ⟦I1= ≪ a≤´x ∧ (´found ⟶ (a<´x ∧ f(´x)=0) ∨ (´y≤a ∧ f(´y)=0))
     ∧ (¬´found ∧ a<´ x ⟶ f(´x)≠0) ≫ ;
   I2= ≪´y≤a+1 ∧ (´found ⟶ (a<´x ∧ f(´x)=0) ∨ (´y≤a ∧ f(´y)=0))
     ∧ (¬´found ∧ ´y≤a ⟶ f(´y)≠0) ≫ ⟧ ⟹
 ||− {|∃ u. f(u)=0|}
 ´turn:=1,, ´found:= False,,
 ´x:=a,, ´y:=a+1 ,,
 COBEGIN {|´I1|}
      WHILE ¬´found
      INV {|´I1|}
      DO {|a≤´x ∧ (´found ⟶ ´y≤a ∧ f(´y)=0) ∧ (a<´x ⟶ f(´x)≠0)|}
         WAIT ´turn=1 END;;
         {|a≤´x ∧ (´found ⟶ ´y≤a ∧ f(´y)=0) ∧ (a<´x ⟶ f(´x)≠0)|}
         ´turn:=2;;
         {|a≤´x ∧ (´found ⟶ ´y≤a ∧ f(´y)=0) ∧ (a<´x ⟶ f(´x)≠0)|}
         ⟨ ´x:=´x+1,,
           IF f(´x)=0 THEN ´found:=True ELSE SKIP FI⟩
      OD;;
      {|´I1 ∧ ´found|}
      ´turn:=2
      {|´I1 ∧ ´found|}
 ||
     {|´I2|}
      WHILE ¬´found
      INV {|´I2|}
      DO {|´y≤a+1 ∧ (´found ⟶ a<´x ∧ f(´x)=0) ∧ (´y≤a ⟶ f(´y)≠0)|}
         WAIT ´turn=2 END;;
         {|´y≤a+1 ∧ (´found ⟶ a<´x ∧ f(´x)=0) ∧ (´y≤a ⟶ f(´y)≠0)|}
         ´turn:=1;;
         {|´y≤a+1 ∧ (´found ⟶ a<´x ∧ f(´x)=0) ∧ (´y≤a ⟶ f(´y)≠0)|}
         ⟨ ´y:=(´y − 1),,
           IF f(´y)=0 THEN ´found:=True ELSE SKIP FI⟩
      OD;;
      {|´I2 ∧ ´found|}
      ´turn:=1
      {|´I2 ∧ ´found|}
 COEND
 {|f(´x)=0 ∨ f(´y)=0|}
⟨proof⟩
```

Easier Version: without AWAIT. Apt and Olderog. page 256:

lemma *Zero-Search-2*:
$[\![I1 = \ll a \leq {}'x \wedge ({}'found \longrightarrow (a < {}'x \wedge f({}'x) = 0) \vee ({}'y \leq a \wedge f({}'y) = 0))$
$\wedge (\neg {}'found \wedge a < {}'x \longrightarrow f({}'x) \neq 0) \gg;$
$I2 = \ll {}'y \leq a+1 \wedge ({}'found \longrightarrow (a < {}'x \wedge f({}'x) = 0) \vee ({}'y \leq a \wedge f({}'y) = 0))$
$\wedge (\neg {}'found \wedge {}'y \leq a \longrightarrow f({}'y) \neq 0) \gg]\!] \Longrightarrow$
$\| - \{\!| \exists u.\ f(u) = 0 |\!\}$
${}'found := False,,$
${}'x := a,,\ {}'y := a+1,,$
$COBEGIN\ \{\!| {}'I1 |\!\}$
$WHILE\ \neg {}'found$
$INV\ \{\!| {}'I1 |\!\}$
$DO\ \{\!| a \leq {}'x \wedge ({}'found \longrightarrow {}'y \leq a \wedge f({}'y) = 0) \wedge (a < {}'x \longrightarrow f({}'x) \neq 0) |\!\}$
$\langle\ {}'x := {}'x+1,,IF\ f({}'x) = 0\ THEN\ \ {}'found := True\ ELSE\ \ SKIP\ FI \rangle$
OD
$\{\!| {}'I1 \wedge {}'found |\!\}$
$\|$
$\{\!| {}'I2 |\!\}$
$WHILE\ \neg {}'found$
$INV\ \{\!| {}'I2 |\!\}$
$DO\ \{\!| {}'y \leq a+1 \wedge ({}'found \longrightarrow a < {}'x \wedge f({}'x) = 0) \wedge ({}'y \leq a \longrightarrow f({}'y) \neq 0) |\!\}$
$\langle\ {}'y := ({}'y - 1),,IF\ f({}'y) = 0\ THEN\ \ {}'found := True\ ELSE\ \ SKIP\ FI \rangle$
OD
$\{\!| {}'I2 \wedge {}'found |\!\}$
$COEND$
$\{\!| f({}'x) = 0 \vee f({}'y) = 0 |\!\}$
$\langle proof \rangle$

1.8.3 Producer/Consumer

Previous lemmas

lemma *nat-lemma2*: $[\![\ b = m*(n::nat) + t;\ a = s*n + u;\ t = u;\ b - a < n\ ]\!] \Longrightarrow m \leq s$
$\langle proof \rangle$

lemma *mod-lemma*: $[\![\ (c::nat) \leq a;\ a < b;\ b - c < n\ ]\!] \Longrightarrow b\ mod\ n \neq a\ mod\ n$
$\langle proof \rangle$

Producer/Consumer Algorithm

record *Producer-consumer* =
ins :: *nat*
outs :: *nat*
li :: *nat*
lj :: *nat*
vx :: *nat*
vy :: *nat*
buffer :: *nat list*
b :: *nat list*

The whole proof takes aprox. 4 minutes.

lemma *Producer-consumer*:

```
  ⟦INIT= «0<length a ∧ 0<length ´buffer ∧ length ´b=length a» ;
    I= «(∀k<´ins. ´outs≤k ⟶ (a ! k) = ´buffer ! (k mod (length ´buffer))) ∧
            ´outs≤´ins ∧ ´ins−´outs≤length ´buffer» ;
    I1= «´I ∧ ´li≤length a» ;
    p1= «´I1 ∧ ´li=´ins» ;
    I2 = «´I ∧ (∀k<´lj. (a ! k)=(´b ! k)) ∧ ´lj≤length a» ;
    p2 = «´I2 ∧ ´lj=´outs» ⟧ ⟹
  ∥− {|´INIT|}
 ´ins:=0,, ´outs:=0,, ´li:=0,, ´lj:=0,,
 COBEGIN {|´p1 ∧ ´INIT|}
   WHILE ´li <length a
     INV {|´p1 ∧ ´INIT|}
   DO {|´p1 ∧ ´INIT ∧ ´li<length a|}
       ´vx:= (a ! ´li);;
     {|´p1 ∧ ´INIT ∧ ´li<length a ∧ ´vx=(a ! ´li)|}
       WAIT ´ins−´outs < length ´buffer END;;
     {|´p1 ∧ ´INIT ∧ ´li<length a ∧ ´vx=(a ! ´li)
       ∧ ´ins−´outs < length ´buffer|}
      ´buffer:=(list-update ´buffer (´ins mod (length ´buffer)) ´vx);;
     {|´p1 ∧ ´INIT ∧ ´li<length a
       ∧ (a ! ´li)=(´buffer ! (´ins mod (length ´buffer)))
       ∧ ´ins−´outs <length ´buffer|}
      ´ins:=´ins+1;;
     {|´I1 ∧ ´INIT ∧ (´li+1)=´ins ∧ ´li<length a|}
      ´li:=´li+1
   OD
  {|´p1 ∧ ´INIT ∧ ´li=length a|}
  ∥
  {|´p2 ∧ ´INIT|}
   WHILE ´lj < length a
     INV {|´p2 ∧ ´INIT|}
   DO {|´p2 ∧ ´lj<length a ∧ ´INIT|}
       WAIT ´outs<´ins END;;
     {|´p2 ∧ ´lj<length a ∧ ´outs<´ins ∧ ´INIT|}
      ´vy:=(´buffer ! (´outs mod (length ´buffer)));;
     {|´p2 ∧ ´lj<length a ∧ ´outs<´ins ∧ ´vy=(a ! ´lj) ∧ ´INIT|}
      ´outs:=´outs+1;;
     {|´I2 ∧ (´lj+1)=´outs ∧ ´lj<length a ∧ ´vy=(a ! ´lj) ∧ ´INIT|}
      ´b:=(list-update ´b ´lj ´vy);;
     {|´I2 ∧ (´lj+1)=´outs ∧ ´lj<length a ∧ (a ! ´lj)=(´b ! ´lj) ∧ ´INIT|}
      ´lj:=´lj+1
   OD
  {|´p2 ∧ ´lj=length a ∧ ´INIT|}
 COEND
 {| ∀k<length a. (a ! k)=(´b ! k)|}
⟨proof⟩
```

1.8.4 Parameterized Examples

Set Elements of an Array to Zero

record *Example1* =
  a :: *nat* ⇒ *nat*

lemma *Example1*:
 ‖− ⦃*True*⦄
   COBEGIN SCHEME [*0*≤*i*<*n*] ⦃*True*⦄ ´*a*:=´*a* (*i*:=*0*) ⦃´*a i*=*0*⦄ *COEND*
  ⦃∀ *i* < *n*. ´*a i* = *0*⦄
⟨*proof*⟩

Same example with lists as auxiliary variables.

record *Example1-list* =
  A :: *nat list*
lemma *Example1-list*:
 ‖− ⦃*n* < *length* ´*A*⦄
   COBEGIN
     SCHEME [*0*≤*i*<*n*] ⦃*n* < *length* ´*A*⦄ ´*A*:=´*A*[*i*:=*0*] ⦃´*A*!*i*=*0*⦄
   COEND
    ⦃∀ *i* < *n*. ´*A*!*i* = *0*⦄
⟨*proof*⟩

Increment a Variable in Parallel

First some lemmas about summation properties.

lemma *Example2-lemma2-aux*: !!*b*. *j*<*n* ⟹
 (∑ *i*=*0*..<*n*. (*b i*::*nat*)) =
 (∑ *i*=*0*..<*j*. *b i*) + *b j* + (∑ *i*=*0*..<*n*−(*Suc j*) . *b* (*Suc j* + *i*))
⟨*proof*⟩

lemma *Example2-lemma2-aux2*:
  !!*b*. *j*≤ *s* ⟹ (∑ *i*::*nat*=*0*..<*j*. (*b* (*s*:=*t*)) *i*) = (∑ *i*=*0*..<*j*. *b i*)
⟨*proof*⟩

lemma *Example2-lemma2*:
!!*b*. ⟦*j*<*n*; *b j*=*0*⟧ ⟹ *Suc* (∑ *i*::*nat*=*0*..<*n*. *b i*)=(∑ *i*=*0*..<*n*. (*b* (*j* := *Suc 0*)) *i*)
⟨*proof*⟩

record *Example2* =
 c :: *nat* ⇒ *nat*
 x :: *nat*

lemma *Example-2*: *0*<*n* ⟹
 ‖− ⦃´*x*=*0* ∧ (∑ *i*=*0*..<*n*. ´*c i*)=*0*⦄
 COBEGIN

$SCHEME\ [0 \le i < n]$
$\{\!|\ \acute{}x=(\sum i=0..<n.\ \acute{}c\ i) \wedge \acute{}c\ i=0|\!\}$
$\langle\ \acute{}x:=\acute{}x+(Suc\ 0),,\ \acute{}c:=\acute{}c\ (i:=(Suc\ 0))\ \rangle$
$\{\!|\ \acute{}x=(\sum i=0..<n.\ \acute{}c\ i) \wedge \acute{}c\ i=(Suc\ 0)|\!\}$
$COEND$
$\{\!|\ \acute{}x=n|\!\}$
$\langle proof \rangle$

end

Chapter 2

Case Study: Single and Multi-Mutator Garbage Collection Algorithms

2.1 Formalization of the Memory

theory *Graph* **imports** *Main* **begin**

datatype *node* = *Black* | *White*

type-synonym *nodes* = *node list*
type-synonym *edge* = *nat* × *nat*
type-synonym *edges* = *edge list*

consts *Roots* :: *nat set*

definition *Proper-Roots* :: *nodes* ⇒ *bool* **where**
$Proper\text{-}Roots\ M \equiv Roots \neq \{\} \wedge Roots \subseteq \{i.\ i<length\ M\}$

definition *Proper-Edges* :: (*nodes* × *edges*) ⇒ *bool* **where**
$Proper\text{-}Edges \equiv (\lambda(M,E).\ \forall\, i<length\ E.\ fst(E!i)<length\ M \wedge snd(E!i)<length\ M)$

definition *BtoW* :: (*edge* × *nodes*) ⇒ *bool* **where**
$BtoW \equiv (\lambda(e,M).\ (M!fst\ e)=Black \wedge (M!snd\ e)\neq Black)$

definition *Blacks* :: *nodes* ⇒ *nat set* **where**
$Blacks\ M \equiv \{i.\ i<length\ M \wedge M!i=Black\}$

definition *Reach* :: *edges* ⇒ *nat set* **where**
$Reach\ E \equiv \{x.\ (\exists\, path.\ 1<length\ path \wedge path!(length\ path - 1)\in Roots \wedge x=path!0$
$\wedge (\forall\, i<length\ path - 1.\ (\exists\, j<length\ E.\ E!j=(path!(i+1),\ path!i))))$
$\vee\ x\in Roots\}$

Reach: the set of reachable nodes is the set of Roots together with the nodes reachable from some Root by a path represented by a list of nodes (at least two since we traverse at least one edge), where two consecutive nodes correspond to an edge in E.

2.1.1 Proofs about Graphs

lemmas *Graph-defs*= *Blacks-def Proper-Roots-def Proper-Edges-def BtoW-def*
declare *Graph-defs* [*simp*]

Graph 1

lemma *Graph1-aux* [*rule-format*]:
$[\![$ $Roots \subseteq Blacks\ M$; $\forall i<length\ E.\ \neg BtoW(E!i,M)]\!]$
$\Longrightarrow 1< length\ path \longrightarrow (path!(length\ path - 1)) \in Roots \longrightarrow$
$(\forall i<length\ path - 1.\ (\exists j.\ j < length\ E \wedge E!j=(path!(Suc\ i),\ path!i)))$
$\longrightarrow M!(path!0) = Black$
⟨*proof*⟩

lemma *Graph1*:
$[\![Roots \subseteq Blacks\ M;\ Proper\text{-}Edges(M,\ E);\ \forall i<length\ E.\ \neg BtoW(E!i,M)\ ]\!]$
$\Longrightarrow Reach\ E \subseteq Blacks\ M$
⟨*proof*⟩

Graph 2

lemma *Ex-first-occurrence* [*rule-format*]:
$P\ (n::nat) \longrightarrow (\exists m.\ P\ m \wedge (\forall i.\ i<m \longrightarrow \neg\ P\ i))$
⟨*proof*⟩

lemma *Compl-lemma*: $(n::nat) \le l \Longrightarrow (\exists m.\ m \le l \wedge n=l - m)$
⟨*proof*⟩

lemma *Ex-last-occurrence*:
$[\![P\ (n::nat);\ n \le l]\!] \Longrightarrow (\exists m.\ P\ (l - m) \wedge (\forall i.\ i<m \longrightarrow \neg P\ (l - i)))$
⟨*proof*⟩

lemma *Graph2*:
$[\![T \in Reach\ E;\ R<length\ E]\!] \Longrightarrow T \in Reach\ (E[R:=(fst(E!R),\ T)])$
⟨*proof*⟩

Graph 3

declare *min.absorb1* [*simp*] *min.absorb2* [*simp*]

lemma *Graph3*:
$[\![\ T \in Reach\ E;\ R<length\ E\ ]\!] \Longrightarrow Reach(E[R:=(fst(E!R),T)]) \subseteq Reach\ E$
⟨*proof*⟩

Graph 4

lemma *Graph4*:
$[\![T \in \textit{Reach } E;\ \textit{Roots}\subseteq\textit{Blacks } M;\ I\le\textit{length } E;\ T<\textit{length } M;\ R<\textit{length } E;$
$\forall\, i<I.\ \neg\textit{BtoW}(E!i,M);\ R<I;\ M!\textit{fst}(E!R)=\textit{Black};\ M!T\neq\textit{Black}]\!] \Longrightarrow$
$(\exists\, r.\ I\le r \wedge r<\textit{length } E \wedge \textit{BtoW}(E[R:=(\textit{fst}(E!R),T)]!r,M))$
$\langle\textit{proof}\rangle$

declare *min.absorb1* [*simp del*] *min.absorb2* [*simp del*]

Graph 5

lemma *Graph5*:
$[\![\ T \in \textit{Reach } E\ ;\ \textit{Roots} \subseteq \textit{Blacks } M;\ \forall\, i<R.\ \neg\textit{BtoW}(E!i,M);\ T<\textit{length } M;$
$R<\textit{length } E;\ M!\textit{fst}(E!R)=\textit{Black};\ M!\textit{snd}(E!R)=\textit{Black};\ M!T \neq \textit{Black}]\!]$
$\Longrightarrow (\exists\, r.\ R<r \wedge r<\textit{length } E \wedge \textit{BtoW}(E[R:=(\textit{fst}(E!R),T)]!r,M))$
$\langle\textit{proof}\rangle$

Other lemmas about graphs

lemma *Graph6*:
$[\![\textit{Proper-Edges}(M,E);\ R<\textit{length } E\ ;\ T<\textit{length } M]\!] \Longrightarrow \textit{Proper-Edges}(M,E[R:=(\textit{fst}(E!R),T)])$
$\langle\textit{proof}\rangle$

lemma *Graph7*:
$[\![\textit{Proper-Edges}(M,E)]\!] \Longrightarrow \textit{Proper-Edges}(M[T:=a],E)$
$\langle\textit{proof}\rangle$

lemma *Graph8*:
$[\![\textit{Proper-Roots}(M)]\!] \Longrightarrow \textit{Proper-Roots}(M[T:=a])$
$\langle\textit{proof}\rangle$

Some specific lemmata for the verification of garbage collection algorithms.

lemma *Graph9*: $j<\textit{length } M \Longrightarrow \textit{Blacks } M\subseteq\textit{Blacks } (M[j := \textit{Black}])$
$\langle\textit{proof}\rangle$

lemma *Graph10* [*rule-format* (*no-asm*)]: $\forall\, i.\ M!i=a \longrightarrow M[i:=a]=M$
$\langle\textit{proof}\rangle$

lemma *Graph11* [*rule-format* (*no-asm*)]:
$[\![\ M!j\neq\textit{Black};j<\textit{length } M]\!] \Longrightarrow \textit{Blacks } M \subset \textit{Blacks } (M[j := \textit{Black}])$
$\langle\textit{proof}\rangle$

lemma *Graph12*: $[\![a\subseteq\textit{Blacks } M;j<\textit{length } M]\!] \Longrightarrow a\subseteq\textit{Blacks } (M[j := \textit{Black}])$
$\langle\textit{proof}\rangle$

lemma *Graph13*: $[\![a\subset \textit{Blacks } M;j<\textit{length } M]\!] \Longrightarrow a \subset \textit{Blacks } (M[j := \textit{Black}])$
$\langle\textit{proof}\rangle$

declare *Graph-defs* [*simp del*]

end

2.2 The Single Mutator Case

theory *Gar-Coll* **imports** *Graph OG-Syntax* **begin**

declare *psubsetE* [*rule del*]

Declaration of variables:

record *gar-coll-state* =
  M :: *nodes*
  E :: *edges*
  bc :: *nat set*
  obc :: *nat set*
  Ma :: *nodes*
  ind :: *nat*
  k :: *nat*
  z :: *bool*

2.2.1 The Mutator

The mutator first redirects an arbitrary edge R from an arbitrary accessible node towards an arbitrary accessible node T. It then colors the new target T black.

We declare the arbitrarily selected node and edge as constants:

consts R :: *nat* T :: *nat*

The following predicate states, given a list of nodes m and a list of edges e, the conditions under which the selected edge R and node T are valid:

definition *Mut-init* :: *gar-coll-state* ⇒ *bool* **where**
  $\mathit{Mut\text{-}init} \equiv \ll T \in \mathit{Reach}\ {}'E \wedge R < \mathit{length}\ {}'E \wedge T < \mathit{length}\ {}'M \gg$

For the mutator we consider two modules, one for each action. An auxiliary variable $'z$ is set to false if the mutator has already redirected an edge but has not yet colored the new target.

definition *Redirect-Edge* :: *gar-coll-state ann-com* **where**
  $\mathit{Redirect\text{-}Edge} \equiv \{\!|\,'\mathit{Mut\text{-}init} \wedge {}'z|\!\}\ \langle {}'E{:=}{}'E[R{:=}(\mathit{fst}({}'E!R),\ T)],,\ {}'z{:=}\ (\neg {}'z)\rangle$

definition *Color-Target* :: *gar-coll-state ann-com* **where**
  $\mathit{Color\text{-}Target} \equiv \{\!|\,'\mathit{Mut\text{-}init} \wedge \neg {}'z|\!\}\ \langle {}'M{:=}{}'M[T{:=}\mathit{Black}],,\ {}'z{:=}\ (\neg {}'z)\rangle$

definition *Mutator* :: *gar-coll-state ann-com* **where**
  Mutator ≡
  $\{\!|\,'\mathit{Mut\text{-}init} \wedge {}'z|\!\}$
  WHILE True INV $\{\!|\,'\mathit{Mut\text{-}init} \wedge {}'z|\!\}$
  DO Redirect-Edge ;; *Color-Target OD*

Correctness of the mutator

lemmas *mutator-defs* = *Mut-init-def Redirect-Edge-def Color-Target-def*

lemma *Redirect-Edge*:
 $\vdash$ *Redirect-Edge pre*(*Color-Target*)
⟨*proof*⟩

lemma *Color-Target*:
 $\vdash$ *Color-Target* {|´*Mut-init* ∧ ´*z*|}
⟨*proof*⟩

lemma *Mutator*:
$\vdash$ *Mutator* {|*False*|}
⟨*proof*⟩

2.2.2 The Collector

A constant *M-init* is used to give ´*Ma* a suitable first value, defined as a list of nodes where only the *Roots* are black.

consts *M-init* :: *nodes*

definition *Proper-M-init* :: *gar-coll-state* ⇒ *bool* **where**
 Proper-M-init ≡ ≪ *Blacks M-init*=*Roots* ∧ *length M-init*=*length* ´*M* ≫

definition *Proper* :: *gar-coll-state* ⇒ *bool* **where**
 Proper ≡ ≪ *Proper-Roots* ´*M* ∧ *Proper-Edges*(´*M*, ´*E*) ∧ ´*Proper-M-init* ≫

definition *Safe* :: *gar-coll-state* ⇒ *bool* **where**
 Safe ≡ ≪ *Reach* ´*E* ⊆ *Blacks* ´*M* ≫

lemmas *collector-defs* = *Proper-M-init-def Proper-def Safe-def*

Blackening the roots

definition *Blacken-Roots* :: *gar-coll-state ann-com* **where**
 Blacken-Roots ≡
 {|´*Proper*|}
 ´*ind*:=*0*;;
 {|´*Proper* ∧ ´*ind*=*0*|}
 WHILE ´*ind*<*length* ´*M*
  INV {|´*Proper* ∧ (∀ *i*<´*ind*. *i* ∈ *Roots* ⟶ ´*M*!*i*=*Black*) ∧ ´*ind*≤*length* ´*M*|}
 DO {|´*Proper* ∧ (∀ *i*<´*ind*. *i* ∈ *Roots* ⟶ ´*M*!*i*=*Black*) ∧ ´*ind*<*length* ´*M*|}
  IF ´*ind*∈*Roots THEN*
   {|´*Proper* ∧ (∀ *i*<´*ind*. *i* ∈ *Roots* ⟶ ´*M*!*i*=*Black*) ∧ ´*ind*<*length* ´*M* ∧ ´*ind*∈*Roots*|}
   ´*M*:=´*M*[´*ind*:=*Black*] *FI*;;
  {|´*Proper* ∧ (∀ *i*<´*ind*+*1*. *i* ∈ *Roots* ⟶ ´*M*!*i*=*Black*) ∧ ´*ind*<*length* ´*M*|}
   ´*ind*:=´*ind*+*1*

OD

lemma *Blacken-Roots*:
$\vdash$ *Blacken-Roots* $\{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M|\!\}$
$\langle proof\rangle$

Propagating black

definition *PBInv* :: *gar-coll-state* $\Rightarrow$ *nat* $\Rightarrow$ *bool* **where**

$PBInv \equiv \ll \lambda ind.\ {}^\prime obc < Blacks\ {}^\prime M \vee (\forall\, i < ind.\ \neg BtoW\ ({}^\prime E!i,\ {}^\prime M) \vee$
$(\neg {}^\prime z \wedge i{=}R \wedge (snd({}^\prime E!R)) = T \wedge (\exists\, r.\ ind \leq r \wedge r < length\ {}^\prime E \wedge BtoW({}^\prime E!r, {}^\prime M))))\gg$

definition *Propagate-Black-aux* :: *gar-coll-state ann-com* **where**

Propagate-Black-aux $\equiv$
$\{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M|\!\}$
${}^\prime ind{:=}0;;$
$\{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime ind{=}0|\!\}$
$WHILE\ {}^\prime ind{<}length\ {}^\prime E$
$INV\ \{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M$
$\wedge\ {}^\prime PBInv\ {}^\prime ind \wedge {}^\prime ind{\leq}length\ {}^\prime E|\!\}$
$DO\ \{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M$
$\wedge\ {}^\prime PBInv\ {}^\prime ind \wedge {}^\prime ind{<}length\ {}^\prime E|\!\}$
$IF\ {}^\prime M!(fst\ ({}^\prime E!{}^\prime ind)) = Black\ THEN$
$\{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M$
$\wedge\ {}^\prime PBInv\ {}^\prime ind \wedge {}^\prime ind{<}length\ {}^\prime E \wedge {}^\prime M!fst({}^\prime E!{}^\prime ind){=}Black|\!\}$
${}^\prime M{:=}{}^\prime M[snd({}^\prime E!{}^\prime ind){:=}Black];;$
$\{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M$
$\wedge\ {}^\prime PBInv\ ({}^\prime ind + 1) \wedge {}^\prime ind{<}length\ {}^\prime E|\!\}$
${}^\prime ind{:=}{}^\prime ind{+}1$
FI
OD

lemma *Propagate-Black-aux*:
$\vdash$ *Propagate-Black-aux*
$\{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M$
$\wedge\ (\ {}^\prime obc < Blacks\ {}^\prime M \vee {}^\prime Safe)|\!\}$
$\langle proof\rangle$

Refining propagating black

definition *Auxk* :: *gar-coll-state* $\Rightarrow$ *bool* **where**

$Auxk \equiv \ll {}^\prime k{<}length\ {}^\prime M \wedge ({}^\prime M!{}^\prime k{\neq}Black \vee \neg BtoW({}^\prime E!{}^\prime ind,\ {}^\prime M) \vee$
${}^\prime obc{<}Blacks\ {}^\prime M \vee (\neg {}^\prime z \wedge {}^\prime ind{=}R \wedge snd({}^\prime E!R){=}T$
$\wedge (\exists\, r.\ {}^\prime ind{<}r \wedge r{<}length\ {}^\prime E \wedge BtoW({}^\prime E!r,\ {}^\prime M))))\gg$

definition *Propagate-Black* :: *gar-coll-state ann-com* **where**

Propagate-Black $\equiv$
$\{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M|\!\}$
${}^\prime ind{:=}0;;$
$\{\!|\,{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime ind{=}0|\!\}$

```
  WHILE ´ind<length ´E
   INV ⦃´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
        ∧ ´PBInv ´ind ∧ ´ind≤length ´E⦄
  DO ⦃´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
      ∧ ´PBInv ´ind ∧ ´ind<length ´E⦄
   IF (´M!(fst (´E!´ind)))=Black THEN
    ⦃´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
     ∧ ´PBInv ´ind ∧ ´ind<length ´E ∧ (´M!fst(´E!´ind))=Black⦄
    ´k:=(snd(´E!´ind));;
    ⦃´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
     ∧ ´PBInv ´ind ∧ ´ind<length ´E ∧ (´M!fst(´E!´ind))=Black
     ∧ ´Auxk⦄
    ⟨´M:=´M[´k:=Black],, ´ind:=´ind+1⟩
   ELSE ⦃´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
         ∧ ´PBInv ´ind ∧ ´ind<length ´E⦄
        ⟨IF (´M!(fst (´E!´ind)))≠Black THEN ´ind:=´ind+1 FI⟩
   FI
  OD
```

lemma *Propagate-Black*:
```
 ⊢ Propagate-Black
  ⦃´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
    ∧ ( ´obc < Blacks ´M ∨ ´Safe)⦄
⟨proof⟩
```

Counting black nodes

definition *CountInv* :: *gar-coll-state* ⇒ *nat* ⇒ *bool* **where**
```
  CountInv ≡ ≪ λind. {i. i<ind ∧ ´Ma!i=Black}⊆´bc ≫
```

definition *Count* :: *gar-coll-state ann-com* **where**
```
  Count ≡
  ⦃´Proper ∧ Roots⊆Blacks ´M
    ∧ ´obc⊆Blacks ´Ma ∧ Blacks ´Ma⊆Blacks ´M ∧ ´bc⊆Blacks ´M
    ∧ length ´Ma=length ´M ∧ (´obc < Blacks ´Ma ∨ ´Safe) ∧ ´bc={}⦄
  ´ind:=0;;
  ⦃´Proper ∧ Roots⊆Blacks ´M
    ∧ ´obc⊆Blacks ´Ma ∧ Blacks ´Ma⊆Blacks ´M ∧ ´bc⊆Blacks ´M
   ∧ length ´Ma=length ´M ∧ (´obc < Blacks ´Ma ∨ ´Safe) ∧ ´bc={}
   ∧ ´ind=0⦄
   WHILE ´ind<length ´M
     INV ⦃´Proper ∧ Roots⊆Blacks ´M
          ∧ ´obc⊆Blacks ´Ma ∧ Blacks ´Ma⊆Blacks ´M ∧ ´bc⊆Blacks ´M
          ∧ length ´Ma=length ´M ∧ ´CountInv ´ind
          ∧ ( ´obc < Blacks ´Ma ∨ ´Safe) ∧ ´ind≤length ´M⦄
   DO ⦃´Proper ∧ Roots⊆Blacks ´M
        ∧ ´obc⊆Blacks ´Ma ∧ Blacks ´Ma⊆Blacks ´M ∧ ´bc⊆Blacks ´M
        ∧ length ´Ma=length ´M ∧ ´CountInv ´ind
        ∧ ( ´obc < Blacks ´Ma ∨ ´Safe) ∧ ´ind<length ´M⦄
```

IF ´M!´ind=Black
  THEN $\{\!|$ *´Proper* $\wedge$ *Roots*$\subseteq$*Blacks ´M*
    $\wedge$ *´obc*$\subseteq$*Blacks ´Ma* $\wedge$ *Blacks ´Ma*$\subseteq$*Blacks ´M* $\wedge$ *´bc*$\subseteq$*Blacks ´M*
    $\wedge$ *length ´Ma=length ´M* $\wedge$ *´CountInv ´ind*
   $\wedge$ (*´obc* < *Blacks ´Ma* $\vee$ *´Safe*) $\wedge$ *´ind<length ´M* $\wedge$ *´M!´ind=Black*$|\!\}$
  ´bc:=insert ´ind ´bc
FI;;
$\{\!|$ *´Proper* $\wedge$ *Roots*$\subseteq$*Blacks ´M*
 $\wedge$ *´obc*$\subseteq$*Blacks ´Ma* $\wedge$ *Blacks ´Ma*$\subseteq$*Blacks ´M* $\wedge$ *´bc*$\subseteq$*Blacks ´M*
 $\wedge$ *length ´Ma=length ´M* $\wedge$ *´CountInv* (*´ind+1*)
 $\wedge$ (*´obc* < *Blacks ´Ma* $\vee$ *´Safe*) $\wedge$ *´ind<length ´M*$|\!\}$
´ind:=´ind+1
OD

lemma *Count*:
 $\vdash$ *Count*
 $\{\!|$ *´Proper* $\wedge$ *Roots*$\subseteq$*Blacks ´M*
 $\wedge$ *´obc*$\subseteq$*Blacks ´Ma* $\wedge$ *Blacks ´Ma*$\subseteq$*´bc* $\wedge$ *´bc*$\subseteq$*Blacks ´M* $\wedge$ *length ´Ma=length ´M*
 $\wedge$ (*´obc* < *Blacks ´Ma* $\vee$ *´Safe*)$|\!\}$
$\langle$*proof*$\rangle$

Appending garbage nodes to the free list

axiomatization *Append-to-free* :: *nat* $\times$ *edges* $\Rightarrow$ *edges*
where
 Append-to-free0: *length* (*Append-to-free* (*i*, *e*)) = *length e* **and**
 Append-to-free1: *Proper-Edges* (*m*, *e*)
   $\Longrightarrow$ *Proper-Edges* (*m*, *Append-to-free*(*i*, *e*)) **and**
 Append-to-free2: *i* $\notin$ *Reach e*
   $\Longrightarrow$ *n* $\in$ *Reach* (*Append-to-free*(*i*, *e*)) = (*n* = *i* $\vee$ *n* $\in$ *Reach e*)

definition *AppendInv* :: *gar-coll-state* $\Rightarrow$ *nat* $\Rightarrow$ *bool* **where**
 AppendInv $\equiv$ $\ll\lambda$*ind*. $\forall$ *i<length ´M*. *ind*$\leq$*i* $\longrightarrow$ *i*$\in$*Reach ´E* $\longrightarrow$ *´M!i=Black*$\gg$

definition *Append* :: *gar-coll-state ann-com* **where**
 Append $\equiv$
 $\{\!|$ *´Proper* $\wedge$ *Roots*$\subseteq$*Blacks ´M* $\wedge$ *´Safe*$|\!\}$
 ´ind:=0;;
 $\{\!|$ *´Proper* $\wedge$ *Roots*$\subseteq$*Blacks ´M* $\wedge$ *´Safe* $\wedge$ *´ind=0*$|\!\}$
  WHILE ´ind<length ´M
   INV $\{\!|$ *´Proper* $\wedge$ *´AppendInv ´ind* $\wedge$ *´ind*$\leq$*length ´M*$|\!\}$
  DO $\{\!|$ *´Proper* $\wedge$ *´AppendInv ´ind* $\wedge$ *´ind<length ´M*$|\!\}$
   IF ´M!´ind=Black THEN
    $\{\!|$ *´Proper* $\wedge$ *´AppendInv ´ind* $\wedge$ *´ind<length ´M* $\wedge$ *´M!´ind=Black*$|\!\}$
    ´M:=´M[´ind:=White]
   ELSE $\{\!|$ *´Proper* $\wedge$ *´AppendInv ´ind* $\wedge$ *´ind*$\notin$*Reach ´E*$|\!\}$
     ´E:=Append-to-free(´ind,´E)
   FI;;

{|´*Proper* ∧ ´*AppendInv* (´*ind*+*1*) ∧ ´*ind*<*length* ´*M*|}
´*ind*:=´*ind*+*1*
OD

lemma *Append*:
⊢ *Append* {|´*Proper*|}
⟨*proof*⟩

Correctness of the Collector

definition *Collector* :: *gar-coll-state ann-com* **where**
Collector ≡
{|´*Proper*|}
WHILE True INV {|´*Proper*|}
DO
Blacken-Roots;;
{|´*Proper* ∧ *Roots*⊆*Blacks* ´*M*|}
´*obc*:={};;
{|´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*={}|}
´*bc*:=*Roots*;;
{|´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*={} ∧ ´*bc*=*Roots*|}
´*Ma*:=*M-init*;;
{|´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*={} ∧ ´*bc*=*Roots* ∧ ´*Ma*=*M-init*|}
WHILE ´*obc*≠´*bc*
INV {|´*Proper* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*Ma* ∧ *Blacks* ´*Ma*⊆´*bc* ∧ ´*bc*⊆*Blacks* ´*M*
∧ *length* ´*Ma*=*length* ´*M* ∧ (´*obc* < *Blacks* ´*Ma* ∨ ´*Safe*)|}
DO {|´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*|}
´*obc*:=´*bc*;;
Propagate-Black;;
{|´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ (´*obc* < *Blacks* ´*M* ∨ ´*Safe*)|}
´*Ma*:=´*M*;;
{|´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*Ma*
∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M* ∧ *length* ´*Ma*=*length* ´*M*
∧ (´*obc* < *Blacks* ´*Ma* ∨ ´*Safe*)|}
´*bc*:={};;
Count
OD;;
Append
OD

lemma *Collector*:
⊢ *Collector* {|*False*|}
⟨*proof*⟩

2.2.3 Interference Freedom

lemmas *modules* = *Redirect-Edge-def Color-Target-def Blacken-Roots-def*
Propagate-Black-def Count-def Append-def

lemmas *Invariants* = *PBInv-def Auxk-def CountInv-def AppendInv-def*
lemmas *abbrev* = *collector-defs mutator-defs Invariants*

lemma *interfree-Blacken-Roots-Redirect-Edge*:
interfree-aux (*Some Blacken-Roots*, {}, *Some Redirect-Edge*)
⟨*proof*⟩

lemma *interfree-Redirect-Edge-Blacken-Roots*:
interfree-aux (*Some Redirect-Edge*, {}, *Some Blacken-Roots*)
⟨*proof*⟩

lemma *interfree-Blacken-Roots-Color-Target*:
interfree-aux (*Some Blacken-Roots*, {}, *Some Color-Target*)
⟨*proof*⟩

lemma *interfree-Color-Target-Blacken-Roots*:
interfree-aux (*Some Color-Target*, {}, *Some Blacken-Roots*)
⟨*proof*⟩

lemma *interfree-Propagate-Black-Redirect-Edge*:
interfree-aux (*Some Propagate-Black*, {}, *Some Redirect-Edge*)
⟨*proof*⟩

lemma *interfree-Redirect-Edge-Propagate-Black*:
interfree-aux (*Some Redirect-Edge*, {}, *Some Propagate-Black*)
⟨*proof*⟩

lemma *interfree-Propagate-Black-Color-Target*:
interfree-aux (*Some Propagate-Black*, {}, *Some Color-Target*)
⟨*proof*⟩

lemma *interfree-Color-Target-Propagate-Black*:
interfree-aux (*Some Color-Target*, {}, *Some Propagate-Black*)
⟨*proof*⟩

lemma *interfree-Count-Redirect-Edge*:
interfree-aux (*Some Count*, {}, *Some Redirect-Edge*)
⟨*proof*⟩

lemma *interfree-Redirect-Edge-Count*:
interfree-aux (*Some Redirect-Edge*, {}, *Some Count*)
⟨*proof*⟩

lemma *interfree-Count-Color-Target*:
interfree-aux (*Some Count*, {}, *Some Color-Target*)
⟨*proof*⟩

lemma *interfree-Color-Target-Count*:
interfree-aux (*Some Color-Target*, {}, *Some Count*)

⟨*proof*⟩

lemma *interfree-Append-Redirect-Edge*:
interfree-aux (*Some Append*, {}, *Some Redirect-Edge*)
⟨*proof*⟩

lemma *interfree-Redirect-Edge-Append*:
interfree-aux (*Some Redirect-Edge*, {}, *Some Append*)
⟨*proof*⟩

lemma *interfree-Append-Color-Target*:
interfree-aux (*Some Append*, {}, *Some Color-Target*)
⟨*proof*⟩

lemma *interfree-Color-Target-Append*:
interfree-aux (*Some Color-Target*, {}, *Some Append*)
⟨*proof*⟩

lemmas *collector-mutator-interfree* =
interfree-Blacken-Roots-Redirect-Edge interfree-Blacken-Roots-Color-Target
interfree-Propagate-Black-Redirect-Edge interfree-Propagate-Black-Color-Target
interfree-Count-Redirect-Edge interfree-Count-Color-Target
interfree-Append-Redirect-Edge interfree-Append-Color-Target
interfree-Redirect-Edge-Blacken-Roots interfree-Color-Target-Blacken-Roots
interfree-Redirect-Edge-Propagate-Black interfree-Color-Target-Propagate-Black
interfree-Redirect-Edge-Count interfree-Color-Target-Count
interfree-Redirect-Edge-Append interfree-Color-Target-Append

Interference freedom Collector-Mutator

lemma *interfree-Collector-Mutator*:
interfree-aux (*Some Collector*, {}, *Some Mutator*)
⟨*proof*⟩

Interference freedom Mutator-Collector

lemma *interfree-Mutator-Collector*:
interfree-aux (*Some Mutator*, {}, *Some Collector*)
⟨*proof*⟩

The Garbage Collection algorithm

In total there are 289 verification conditions.

lemma *Gar-Coll*:
‖− {|´*Proper* ∧ ´*Mut-init* ∧ ´*z*|}
COBEGIN
Collector
{|*False*|}
‖

Mutator
$\{\!|False|\!\}$
COEND
$\{\!|False|\!\}$
$\langle proof \rangle$

end

2.3 The Multi-Mutator Case

theory *Mul-Gar-Coll* **imports** *Graph OG-Syntax* **begin**

The full theory takes aprox. 18 minutes.

record *mut* =
Z :: *bool*
R :: *nat*
T :: *nat*

Declaration of variables:

record *mul-gar-coll-state* =
M :: *nodes*
E :: *edges*
bc :: *nat set*
obc :: *nat set*
Ma :: *nodes*
ind :: *nat*
k :: *nat*
q :: *nat*
l :: *nat*
Muts :: *mut list*

2.3.1 The Mutators

definition *Mul-mut-init* :: *mul-gar-coll-state* ⇒ *nat* ⇒ *bool* **where**
$Mul\text{-}mut\text{-}init \equiv \ll \lambda n.\ n{=}length\ {}´Muts \wedge (\forall\, i{<}n.\ R\ ({}´Muts!i){<}length\ {}´E$
$\wedge\ T\ ({}´Muts!i){<}length\ {}´M) \gg$

definition *Mul-Redirect-Edge* :: *nat* ⇒ *nat* ⇒ *mul-gar-coll-state ann-com* **where**
$Mul\text{-}Redirect\text{-}Edge\ j\ n \equiv$
$\{\!|{}´Mul\text{-}mut\text{-}init\ n \wedge Z\ ({}´Muts!j)|\!\}$
$\langle IF\ T({}´Muts!j) \in Reach\ {}´E\ THEN$
${}´E{:=}\ {}´E[R\ ({}´Muts!j){:=}\ (fst\ ({}´E!R({}´Muts!j)),\ T\ ({}´Muts!j))]\ FI,,$
${}´Muts{:=}\ {}´Muts[j{:=}\ ({}´Muts!j)\ (|Z{:=}False|)]\rangle$

definition *Mul-Color-Target* :: *nat* ⇒ *nat* ⇒ *mul-gar-coll-state ann-com* **where**
$Mul\text{-}Color\text{-}Target\ j\ n \equiv$
$\{\!|{}´Mul\text{-}mut\text{-}init\ n \wedge \neg\ Z\ ({}´Muts!j)|\!\}$
$\langle {}´M{:=}{}´M[T\ ({}´Muts!j){:=}Black],,\ {}´Muts{:=}{}´Muts[j{:=}\ ({}´Muts!j)\ (|Z{:=}True|)]\rangle$

definition *Mul-Mutator* :: *nat* ⇒ *nat* ⇒ *mul-gar-coll-state ann-com* **where**
Mul-Mutator j n ≡
{|´*Mul-mut-init n* ∧ *Z* (´*Muts*!*j*)|}
WHILE True
INV {|´*Mul-mut-init n* ∧ *Z* (´*Muts*!*j*)|}
DO Mul-Redirect-Edge j n ;;
Mul-Color-Target j n
OD

lemmas *mul-mutator-defs* = *Mul-mut-init-def Mul-Redirect-Edge-def Mul-Color-Target-def*

Correctness of the proof outline of one mutator

lemma *Mul-Redirect-Edge*: $0 \le j \wedge j < n \Longrightarrow$
⊢ *Mul-Redirect-Edge j n*
pre(*Mul-Color-Target j n*)
⟨*proof*⟩

lemma *Mul-Color-Target*: $0 \le j \wedge j < n \Longrightarrow$
⊢ *Mul-Color-Target j n*
{|´*Mul-mut-init n* ∧ *Z* (´*Muts*!*j*)|}
⟨*proof*⟩

lemma *Mul-Mutator*: $0 \le j \wedge j < n \Longrightarrow$
⊢ *Mul-Mutator j n* {|*False*|}
⟨*proof*⟩

Interference freedom between mutators

lemma *Mul-interfree-Redirect-Edge-Redirect-Edge*:
⟦$0 \le i$; $i < n$; $0 \le j$; $j < n$; $i \neq j$⟧ ⟹
interfree-aux (*Some* (*Mul-Redirect-Edge i n*),{}, *Some*(*Mul-Redirect-Edge j n*))
⟨*proof*⟩

lemma *Mul-interfree-Redirect-Edge-Color-Target*:
⟦$0 \le i$; $i < n$; $0 \le j$; $j < n$; $i \neq j$⟧ ⟹
interfree-aux (*Some*(*Mul-Redirect-Edge i n*),{},*Some*(*Mul-Color-Target j n*))
⟨*proof*⟩

lemma *Mul-interfree-Color-Target-Redirect-Edge*:
⟦$0 \le i$; $i < n$; $0 \le j$; $j < n$; $i \neq j$⟧ ⟹
interfree-aux (*Some*(*Mul-Color-Target i n*),{},*Some*(*Mul-Redirect-Edge j n*))
⟨*proof*⟩

lemma *Mul-interfree-Color-Target-Color-Target*:
⟦$0 \le i$; $i < n$; $0 \le j$; $j < n$; $i \neq j$⟧ ⟹
interfree-aux (*Some*(*Mul-Color-Target i n*),{},*Some*(*Mul-Color-Target j n*))
⟨*proof*⟩

lemmas *mul-mutator-interfree* =
Mul-interfree-Redirect-Edge-Redirect-Edge Mul-interfree-Redirect-Edge-Color-Target
Mul-interfree-Color-Target-Redirect-Edge Mul-interfree-Color-Target-Color-Target

lemma *Mul-interfree-Mutator-Mutator*: $[\![i < n;\ j < n;\ i \neq j]\!] \Longrightarrow$
interfree-aux (*Some* (*Mul-Mutator i n*), {}, *Some* (*Mul-Mutator j n*))
⟨*proof*⟩

Modular Parameterized Mutators

lemma *Mul-Parameterized-Mutators*: $0<n \Longrightarrow$
$\|-$ ⦃´*Mul-mut-init n* ∧ (∀ *i*<*n*. *Z* (´*Muts*!*i*))⦄
COBEGIN
SCHEME [*0*≤ *j*< *n*]
Mul-Mutator j n
⦃*False*⦄
COEND
⦃*False*⦄
⟨*proof*⟩

2.3.2 The Collector

definition *Queue* :: *mul-gar-coll-state* ⇒ *nat* **where**
Queue ≡ ≪ *length* (*filter* (λ*i*. ¬ *Z i* ∧ ´*M*!(*T i*) ≠ *Black*) ´*Muts*) ≫

consts *M-init* :: *nodes*

definition *Proper-M-init* :: *mul-gar-coll-state* ⇒ *bool* **where**
Proper-M-init ≡ ≪ *Blacks M-init*=*Roots* ∧ *length M-init*=*length* ´*M* ≫

definition *Mul-Proper* :: *mul-gar-coll-state* ⇒ *nat* ⇒ *bool* **where**
Mul-Proper ≡ ≪ λ*n*. *Proper-Roots* ´*M* ∧ *Proper-Edges* (´*M*, ´*E*) ∧ ´*Proper-M-init*
∧ *n*=*length* ´*Muts* ≫

definition *Safe* :: *mul-gar-coll-state* ⇒ *bool* **where**
Safe ≡ ≪ *Reach* ´*E* ⊆ *Blacks* ´*M* ≫

lemmas *mul-collector-defs* = *Proper-M-init-def Mul-Proper-def Safe-def*

Blackening Roots

definition *Mul-Blacken-Roots* :: *nat* ⇒ *mul-gar-coll-state ann-com* **where**
Mul-Blacken-Roots n ≡
⦃´*Mul-Proper n*⦄
´*ind*:=*0*;;
⦃´*Mul-Proper n* ∧ ´*ind*=*0*⦄
WHILE ´*ind*<*length* ´*M*
INV ⦃´*Mul-Proper n* ∧ (∀ *i*<´*ind*. *i*∈*Roots* ⟶ ´*M*!*i*=*Black*) ∧ ´*ind*≤*length*
´*M*⦄
DO ⦃´*Mul-Proper n* ∧ (∀ *i*<´*ind*. *i*∈*Roots* ⟶ ´*M*!*i*=*Black*) ∧ ´*ind*<*length* ´*M*⦄

IF ´ind∈Roots THEN
{| ´*Mul-Proper n* ∧ (∀ *i*<´*ind*. *i*∈*Roots* ⟶ ´*M*!*i*=*Black*) ∧ ´*ind*<*length* ´*M* ∧ ´*ind*∈*Roots* |}
´*M*:=´*M*[´*ind*:=*Black*] *FI*;;
{| ´*Mul-Proper n* ∧ (∀ *i*<´*ind*+*1*. *i*∈*Roots* ⟶ ´*M*!*i*=*Black*) ∧ ´*ind*<*length* ´*M* |}
´*ind*:=´*ind*+*1*
OD

lemma *Mul-Blacken-Roots*:
⊢ *Mul-Blacken-Roots n*
{| ´*Mul-Proper n* ∧ *Roots* ⊆ *Blacks* ´*M* |}
⟨*proof*⟩

Propagating Black

definition *Mul-PBInv* :: *mul-gar-coll-state* ⇒ *bool* **where**
Mul-PBInv ≡ ≪´*Safe* ∨ ´*obc*⊂*Blacks* ´*M* ∨ ´*l*<´*Queue*
∨ (∀ *i*<´*ind*. ¬*BtoW*(´*E*!*i*,´*M*)) ∧ ´*l*≤´*Queue*≫

definition *Mul-Auxk* :: *mul-gar-coll-state* ⇒ *bool* **where**
Mul-Auxk ≡ ≪´*l*<´*Queue* ∨ ´*M*!´*k*≠*Black* ∨ ¬*BtoW*(´*E*!´*ind*, ´*M*) ∨ ´*obc*⊂*Blacks* ´*M*≫

definition *Mul-Propagate-Black* :: *nat* ⇒ *mul-gar-coll-state ann-com* **where**
Mul-Propagate-Black n ≡
{| ´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ (´*Safe* ∨ ´*l*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*) |}
´*ind*:=*0*;;
{| ´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*M* ∧ *Blacks* ´*M*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ (´*Safe* ∨ ´*l*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*) ∧ ´*ind*=*0* |}
WHILE ´*ind*<*length* ´*E*
INV {| ´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ ´*Mul-PBInv* ∧ ´*ind*≤*length* ´*E* |}
DO {| ´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ ´*Mul-PBInv* ∧ ´*ind*<*length* ´*E* |}
IF ´*M*!(*fst* (´*E*!´*ind*))=*Black THEN*
{| ´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ ´*Mul-PBInv* ∧ (´*M*!*fst*(´*E*!´*ind*))=*Black* ∧ ´*ind*<*length* ´*E* |}
´*k*:=*snd*(´*E*!´*ind*);;
{| ´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*M* ∨ ´*l*<´*Queue* ∨ (∀ *i*<´*ind*. ¬*BtoW*(´*E*!*i*,´*M*))
∧ ´*l*≤´*Queue* ∧ ´*Mul-Auxk*) ∧ ´*k*<*length* ´*M* ∧ ´*M*!*fst*(´*E*!´*ind*)=*Black*
∧ ´*ind*<*length* ´*E* |}

$\langle$ ´*M*:=´*M*[´*k*:=*Black*],,´*ind*:=´*ind*+*1*$\rangle$
ELSE {|´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ ´*Mul-PBInv* ∧ ´*ind*<*length* ´*E*|}
$\langle$*IF* ´*M*!(*fst* (´*E*!´*ind*))≠*Black THEN* ´*ind*:=´*ind*+*1 FI*$\rangle$ *FI*
OD

lemma *Mul-Propagate-Black*:
⊢ *Mul-Propagate-Black n*
{|´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*M* ∨ ´*l*<´*Queue* ∧ (´*l*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*))|}
$\langle$*proof*$\rangle$

Counting Black Nodes

definition *Mul-CountInv* :: *mul-gar-coll-state* ⇒ *nat* ⇒ *bool* **where**
Mul-CountInv ≡ « λ*ind*. {*i*. *i*<*ind* ∧ ´*Ma*!*i*=*Black*}⊆´*bc* »

definition *Mul-Count* :: *nat* ⇒ *mul-gar-coll-state ann-com* **where**
Mul-Count n ≡
{|´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*Ma* ∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ *length* ´*Ma*=*length* ´*M*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*))
∧ ´*q*<*n*+*1* ∧ ´*bc*={}|}
´*ind*:=*0*;;
{|´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*Ma* ∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ *length* ´*Ma*=*length* ´*M*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*))
∧ ´*q*<*n*+*1* ∧ ´*bc*={} ∧ ´*ind*=*0*|}
WHILE ´*ind*<*length* ´*M*
INV {|´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*Ma* ∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ *length* ´*Ma*=*length* ´*M* ∧ ´*Mul-CountInv* ´*ind*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*))
∧ ´*q*<*n*+*1* ∧ ´*ind*≤*length* ´*M*|}
DO {|´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*Ma* ∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ *length* ´*Ma*=*length* ´*M* ∧ ´*Mul-CountInv* ´*ind*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*))
∧ ´*q*<*n*+*1* ∧ ´*ind*<*length* ´*M*|}
IF ´*M*!´*ind*=*Black*
THEN {|´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*Ma* ∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ *length* ´*Ma*=*length* ´*M* ∧ ´*Mul-CountInv* ´*ind*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks*

$\acute{}M$))
$\land$ $\acute{}q$<n+1 $\land$ $\acute{}ind$<length $\acute{}M$ $\land$ $\acute{}M!\acute{}ind$=Black⦄
$\acute{}bc$:=insert $\acute{}ind$ $\acute{}bc$
FI;;
⦃$\acute{}$Mul-Proper n $\land$ Roots⊆Blacks $\acute{}M$
$\land$ $\acute{}obc$⊆Blacks $\acute{}Ma$ $\land$ Blacks $\acute{}Ma$⊆Blacks $\acute{}M$ $\land$ $\acute{}bc$⊆Blacks $\acute{}M$
$\land$ length $\acute{}Ma$=length $\acute{}M$ $\land$ $\acute{}$Mul-CountInv ($\acute{}ind$+1)
$\land$ ($\acute{}Safe$ $\lor$ $\acute{}obc$⊂Blacks $\acute{}Ma$ $\lor$ $\acute{}l$<$\acute{}q$ $\land$ ($\acute{}q$≤$\acute{}Queue$ $\lor$ $\acute{}obc$⊂Blacks $\acute{}M$))
$\land$ $\acute{}q$<n+1 $\land$ $\acute{}ind$<length $\acute{}M$⦄
$\acute{}ind$:=$\acute{}ind$+1
OD

lemma *Mul-Count*:
⊢ Mul-Count n
⦃$\acute{}$Mul-Proper n $\land$ Roots⊆Blacks $\acute{}M$
$\land$ $\acute{}obc$⊆Blacks $\acute{}Ma$ $\land$ Blacks $\acute{}Ma$⊆Blacks $\acute{}M$ $\land$ $\acute{}bc$⊆Blacks $\acute{}M$
$\land$ length $\acute{}Ma$=length $\acute{}M$ $\land$ Blacks $\acute{}Ma$⊆$\acute{}bc$
$\land$ ($\acute{}Safe$ $\lor$ $\acute{}obc$⊂Blacks $\acute{}Ma$ $\lor$ $\acute{}l$<$\acute{}q$ $\land$ ($\acute{}q$≤$\acute{}Queue$ $\lor$ $\acute{}obc$⊂Blacks $\acute{}M$))
$\land$ $\acute{}q$<n+1⦄
⟨proof⟩

Appending garbage nodes to the free list

axiomatization *Append-to-free* :: nat × edges ⇒ edges
where
Append-to-free0: length (Append-to-free (i, e)) = length e **and**
Append-to-free1: Proper-Edges (m, e)
⟹ Proper-Edges (m, Append-to-free(i, e)) **and**
Append-to-free2: i ∉ Reach e
⟹ n ∈ Reach (Append-to-free(i, e)) = (n = i ∨ n ∈ Reach e)

definition *Mul-AppendInv* :: mul-gar-coll-state ⇒ nat ⇒ bool **where**
Mul-AppendInv ≡ « λind. (∀ i. ind≤i ⟶ i<length $\acute{}M$ ⟶ i∈Reach $\acute{}E$ ⟶ $\acute{}M$!i=Black)»

definition *Mul-Append* :: nat ⇒ mul-gar-coll-state ann-com **where**
Mul-Append n ≡
⦃$\acute{}$Mul-Proper n $\land$ Roots⊆Blacks $\acute{}M$ $\land$ $\acute{}Safe$⦄
$\acute{}ind$:=0;;
⦃$\acute{}$Mul-Proper n $\land$ Roots⊆Blacks $\acute{}M$ $\land$ $\acute{}Safe$ $\land$ $\acute{}ind$=0⦄
WHILE $\acute{}ind$<length $\acute{}M$
INV ⦃$\acute{}$Mul-Proper n $\land$ $\acute{}$Mul-AppendInv $\acute{}ind$ $\land$ $\acute{}ind$≤length $\acute{}M$⦄
DO ⦃$\acute{}$Mul-Proper n $\land$ $\acute{}$Mul-AppendInv $\acute{}ind$ $\land$ $\acute{}ind$<length $\acute{}M$⦄
IF $\acute{}M!\acute{}ind$=Black THEN
⦃$\acute{}$Mul-Proper n $\land$ $\acute{}$Mul-AppendInv $\acute{}ind$ $\land$ $\acute{}ind$<length $\acute{}M$ $\land$ $\acute{}M!\acute{}ind$=Black⦄
$\acute{}M$:=$\acute{}M$[$\acute{}ind$:=White]
ELSE
⦃$\acute{}$Mul-Proper n $\land$ $\acute{}$Mul-AppendInv $\acute{}ind$ $\land$ $\acute{}ind$<length $\acute{}M$ $\land$ $\acute{}ind$∉Reach $\acute{}E$⦄

´E:=Append-to-free(´ind,´E)
FI;;
{|´Mul-Proper n ∧ ´Mul-AppendInv (´ind+1) ∧ ´ind<length ´M|}
´ind:=´ind+1
OD

lemma *Mul-Append*:
⊢ *Mul-Append n*
{|´Mul-Proper n|}
⟨*proof*⟩

Collector

definition *Mul-Collector* :: *nat* ⇒ *mul-gar-coll-state ann-com* **where**
Mul-Collector n ≡
{|´Mul-Proper n|}
WHILE True INV {|´Mul-Proper n|}
DO
Mul-Blacken-Roots n ;;
{|´Mul-Proper n ∧ Roots⊆Blacks ´M|}
´obc:={};;
{|´Mul-Proper n ∧ Roots⊆Blacks ´M ∧ ´obc={}|}
´bc:=Roots;;
{|´Mul-Proper n ∧ Roots⊆Blacks ´M ∧ ´obc={} ∧ ´bc=Roots|}
´l:=0;;
{|´Mul-Proper n ∧ Roots⊆Blacks ´M ∧ ´obc={} ∧ ´bc=Roots ∧ ´l=0|}
WHILE ´l<n+1
INV {|´Mul-Proper n ∧ Roots⊆Blacks ´M ∧ ´bc⊆Blacks ´M ∧
(´Safe ∨ (´l≤´Queue ∨ ´bc⊂Blacks ´M) ∧ ´l<n+1)|}
DO {|´Mul-Proper n ∧ Roots⊆Blacks ´M ∧ ´bc⊆Blacks ´M
∧ (´Safe ∨ ´l≤´Queue ∨ ´bc⊂Blacks ´M)|}
´obc:=´bc;;
Mul-Propagate-Black n;;
{|´Mul-Proper n ∧ Roots⊆Blacks ´M
∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
∧ (´Safe ∨ ´obc⊂Blacks ´M ∨ ´l<´Queue
∧ (´l≤´Queue ∨ ´obc⊂Blacks ´M))|}
´bc:={};;
{|´Mul-Proper n ∧ Roots⊆Blacks ´M
∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
∧ (´Safe ∨ ´obc⊂Blacks ´M ∨ ´l<´Queue
∧ (´l≤´Queue ∨ ´obc⊂Blacks ´M)) ∧ ´bc={}|}
⟨ ´Ma:=´M,, ´q:=´Queue ⟩;;
Mul-Count n;;
{|´Mul-Proper n ∧ Roots⊆Blacks ´M
∧ ´obc⊆Blacks ´Ma ∧ Blacks ´Ma⊆Blacks ´M ∧ ´bc⊆Blacks ´M
∧ length ´Ma=length ´M ∧ Blacks ´Ma⊆´bc
∧ (´Safe ∨ ´obc⊂Blacks ´Ma ∨ ´l<´q ∧ (´q≤´Queue ∨ ´obc⊂Blacks ´M))
∧ ´q<n+1|}

IF ´obc=´bc THEN
⦃´*Mul-Proper n* ∧ *Roots*⊆*Blacks ´M*
∧ ´*obc*⊆*Blacks ´Ma* ∧ *Blacks ´Ma*⊆*Blacks ´M* ∧ ´*bc*⊆*Blacks ´M*
∧ *length ´Ma*=*length ´M* ∧ *Blacks ´Ma*⊆´*bc*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks ´Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks ´M*))
∧ ´*q*<*n*+*1* ∧ ´*obc*=´*bc*⦄
´*l*:=´*l*+*1*
ELSE ⦃´*Mul-Proper n* ∧ *Roots*⊆*Blacks ´M*
∧ ´*obc*⊆*Blacks ´Ma* ∧ *Blacks ´Ma*⊆*Blacks ´M* ∧ ´*bc*⊆*Blacks ´M*
∧ *length ´Ma*=*length ´M* ∧ *Blacks ´Ma*⊆´*bc*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks ´Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks*
´*M*))
∧ ´*q*<*n*+*1* ∧ ´*obc*≠´*bc*⦄
´*l*:=*0* *FI*
OD;;
Mul-Append n
OD

lemmas *mul-modules* = *Mul-Redirect-Edge-def Mul-Color-Target-def*
Mul-Blacken-Roots-def Mul-Propagate-Black-def
Mul-Count-def Mul-Append-def

lemma *Mul-Collector*:
⊢ *Mul-Collector n*
⦃*False*⦄
⟨*proof*⟩

2.3.3 Interference Freedom

lemma *le-length-filter-update*[*rule-format*]:
∀ *i*. (¬*P* (*list*!*i*) ∨ *P j*) ∧ *i*<*length list*
⟶ *length*(*filter P list*) ≤ *length*(*filter P* (*list*[*i*:=*j*]))
⟨*proof*⟩

lemma *less-length-filter-update* [*rule-format*]:
∀ *i*. *P j* ∧ ¬(*P* (*list*!*i*)) ∧ *i*<*length list*
⟶ *length*(*filter P list*) < *length*(*filter P* (*list*[*i*:=*j*]))
⟨*proof*⟩

lemma *Mul-interfree-Blacken-Roots-Redirect-Edge*: ⟦*0*≤*j*; *j*<*n*⟧ ⟹
interfree-aux (*Some*(*Mul-Blacken-Roots n*),{},*Some*(*Mul-Redirect-Edge j n*))
⟨*proof*⟩

lemma *Mul-interfree-Redirect-Edge-Blacken-Roots*: ⟦*0*≤*j*; *j*<*n*⟧⟹
interfree-aux (*Some*(*Mul-Redirect-Edge j n*),{},*Some* (*Mul-Blacken-Roots n*))
⟨*proof*⟩

lemma *Mul-interfree-Blacken-Roots-Color-Target*: ⟦*0*≤*j*; *j*<*n*⟧⟹
interfree-aux (*Some*(*Mul-Blacken-Roots n*),{},*Some* (*Mul-Color-Target j n*))

⟨*proof*⟩

lemma *Mul-interfree-Color-Target-Blacken-Roots*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Color-Target j n*),{},*Some* (*Mul-Blacken-Roots n*))
⟨*proof*⟩

lemma *Mul-interfree-Propagate-Black-Redirect-Edge*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Propagate-Black n*),{},*Some* (*Mul-Redirect-Edge j n*))
⟨*proof*⟩

lemma *Mul-interfree-Redirect-Edge-Propagate-Black*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Redirect-Edge j n*),{},*Some* (*Mul-Propagate-Black n*))
⟨*proof*⟩

lemma *Mul-interfree-Propagate-Black-Color-Target*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Propagate-Black n*),{},*Some* (*Mul-Color-Target j n*))
⟨*proof*⟩

lemma *Mul-interfree-Color-Target-Propagate-Black*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Color-Target j n*),{},*Some*(*Mul-Propagate-Black n*))
⟨*proof*⟩

lemma *Mul-interfree-Count-Redirect-Edge*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Count n*),{},*Some*(*Mul-Redirect-Edge j n*))
⟨*proof*⟩

lemma *Mul-interfree-Redirect-Edge-Count*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Redirect-Edge j n*),{},*Some*(*Mul-Count n*))
⟨*proof*⟩

lemma *Mul-interfree-Count-Color-Target*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Count n*),{},*Some*(*Mul-Color-Target j n*))
⟨*proof*⟩

lemma *Mul-interfree-Color-Target-Count*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Color-Target j n*),{}, *Some*(*Mul-Count n*))
⟨*proof*⟩

lemma *Mul-interfree-Append-Redirect-Edge*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Append n*),{}, *Some*(*Mul-Redirect-Edge j n*))
⟨*proof*⟩

lemma *Mul-interfree-Redirect-Edge-Append*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Redirect-Edge j n*),{},*Some*(*Mul-Append n*))
⟨*proof*⟩

lemma *Mul-interfree-Append-Color-Target*: $[\![0 \le j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Append n*),{}, *Some*(*Mul-Color-Target j n*))
⟨*proof*⟩

lemma *Mul-interfree-Color-Target-Append*: $[\![0 \leq j;\ j<n]\!] \Longrightarrow$
interfree-aux (*Some*(*Mul-Color-Target j n*),{}, *Some*(*Mul-Append n*))
⟨*proof*⟩

Interference freedom Collector-Mutator

lemmas *mul-collector-mutator-interfree* =
Mul-interfree-Blacken-Roots-Redirect-Edge Mul-interfree-Blacken-Roots-Color-Target
Mul-interfree-Propagate-Black-Redirect-Edge Mul-interfree-Propagate-Black-Color-Target
Mul-interfree-Count-Redirect-Edge Mul-interfree-Count-Color-Target
Mul-interfree-Append-Redirect-Edge Mul-interfree-Append-Color-Target
Mul-interfree-Redirect-Edge-Blacken-Roots Mul-interfree-Color-Target-Blacken-Roots
Mul-interfree-Redirect-Edge-Propagate-Black Mul-interfree-Color-Target-Propagate-Black
Mul-interfree-Redirect-Edge-Count Mul-interfree-Color-Target-Count
Mul-interfree-Redirect-Edge-Append Mul-interfree-Color-Target-Append

lemma *Mul-interfree-Collector-Mutator*: $j<n \Longrightarrow$
interfree-aux (*Some* (*Mul-Collector n*), {}, *Some* (*Mul-Mutator j n*))
⟨*proof*⟩

Interference freedom Mutator-Collector

lemma *Mul-interfree-Mutator-Collector*: $j < n \Longrightarrow$
interfree-aux (*Some* (*Mul-Mutator j n*), {}, *Some* (*Mul-Collector n*))
⟨*proof*⟩

The Multi-Mutator Garbage Collection Algorithm

The total number of verification conditions is 328

lemma *Mul-Gar-Coll*:
$\Vert-$ {|´*Mul-Proper n* ∧ ´*Mul-mut-init n* ∧ (∀ *i*<*n*. *Z* (´*Muts*!*i*))|}
COBEGIN
Mul-Collector n
{|*False*|}
$\Vert$
SCHEME [$0\leq j< n$]
Mul-Mutator j n
{|*False*|}
COEND
{|*False*|}
⟨*proof*⟩

end

Chapter 3

The Rely-Guarantee Method

3.1 Abstract Syntax

theory *RG-Com* **imports** *Main* **begin**

Semantics of assertions and boolean expressions (bexp) as sets of states. Syntax of commands *com* and parallel commands *par-com*.

type-synonym *'a bexp* = *'a set*

datatype *'a com* =
  Basic 'a ⇒'a
  | *Seq 'a com 'a com*
  | *Cond 'a bexp 'a com 'a com*
  | *While 'a bexp 'a com*
  | *Await 'a bexp 'a com*

type-synonym *'a par-com* = *'a com option list*

end

3.2 Operational Semantics

theory *RG-Tran*
imports *RG-Com*
begin

3.2.1 Semantics of Component Programs

Environment transitions

type-synonym *'a conf* = ((*'a com*) *option*) × *'a*

inductive-set
  etran :: (*'a conf* × *'a conf*) *set*
  and *etran'* :: *'a conf* ⇒ *'a conf* ⇒ *bool* (- −*e*→ - [*81*,*81*] *80*)

where
$P -e\rightarrow Q \equiv (P,Q) \in etran$
| Env: $(P,\ s) -e\rightarrow (P,\ t)$

lemma $etranE$: $c -e\rightarrow c' \Longrightarrow (\bigwedge P\ s\ t.\ c = (P,\ s) \Longrightarrow c' = (P,\ t) \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

Component transitions

inductive-set
$ctran :: ('a\ conf \times 'a\ conf)\ set$
and $ctran' :: 'a\ conf \Rightarrow 'a\ conf \Rightarrow bool$ $(\text{-} -c\rightarrow \text{-}\ [81,81]\ 80)$
and $ctrans :: 'a\ conf \Rightarrow 'a\ conf \Rightarrow bool$ $(\text{-} -c{*}\rightarrow \text{-}\ [81,81]\ 80)$
where
$P -c\rightarrow Q \equiv (P,Q) \in ctran$
| $P -c{*}\rightarrow Q \equiv (P,Q) \in ctran^*$

| $Basic$: $(Some(Basic\ f),\ s) -c\rightarrow (None,\ f\ s)$

| $Seq1$: $(Some\ P0,\ s) -c\rightarrow (None,\ t) \Longrightarrow (Some(Seq\ P0\ P1),\ s) -c\rightarrow (Some\ P1,\ t)$

| $Seq2$: $(Some\ P0,\ s) -c\rightarrow (Some\ P2,\ t) \Longrightarrow (Some(Seq\ P0\ P1),\ s) -c\rightarrow (Some(Seq\ P2\ P1),\ t)$

| $CondT$: $s{\in}b \Longrightarrow (Some(Cond\ b\ P1\ P2),\ s) -c\rightarrow (Some\ P1,\ s)$
| $CondF$: $s{\notin}b \Longrightarrow (Some(Cond\ b\ P1\ P2),\ s) -c\rightarrow (Some\ P2,\ s)$

| $WhileF$: $s{\notin}b \Longrightarrow (Some(While\ b\ P),\ s) -c\rightarrow (None,\ s)$
| $WhileT$: $s{\in}b \Longrightarrow (Some(While\ b\ P),\ s) -c\rightarrow (Some(Seq\ P\ (While\ b\ P)),\ s)$

| $Await$: $[\![s{\in}b;\ (Some\ P,\ s) -c{*}\rightarrow (None,\ t)]\!] \Longrightarrow (Some(Await\ b\ P),\ s) -c\rightarrow (None,\ t)$

monos *rtrancl-mono*

3.2.2 Semantics of Parallel Programs

type-synonym $'a\ par\text{-}conf = ('a\ par\text{-}com) \times 'a$

inductive-set
$par\text{-}etran :: ('a\ par\text{-}conf \times 'a\ par\text{-}conf)\ set$
and $par\text{-}etran' :: ['a\ par\text{-}conf,'a\ par\text{-}conf] \Rightarrow bool$ $(\text{-} -pe\rightarrow \text{-}\ [81,81]\ 80)$
where
$P -pe\rightarrow Q \equiv (P,Q) \in par\text{-}etran$
| $ParEnv$: $(Ps,\ s) -pe\rightarrow (Ps,\ t)$

inductive-set
$par\text{-}ctran :: ('a\ par\text{-}conf \times 'a\ par\text{-}conf)\ set$
and $par\text{-}ctran' :: ['a\ par\text{-}conf,'a\ par\text{-}conf] \Rightarrow bool$ $(\text{-} -pc\rightarrow \text{-}\ [81,81]\ 80)$

where
  $P\ {-pc\rightarrow}\ Q \equiv (P,Q) \in$ *par-ctran*
| *ParComp*: $[\![i<length\ Ps;\ (Ps!i,\ s)\ {-c\rightarrow}\ (r,\ t)]\!] \Longrightarrow (Ps,\ s)\ {-pc\rightarrow}\ (Ps[i{:=}r],\ t)$

lemma *par-ctranE*: $c\ {-pc\rightarrow}\ c' \Longrightarrow$
  $(\bigwedge i\ Ps\ s\ r\ t.\ c = (Ps,\ s) \Longrightarrow c' = (Ps[i := r],\ t) \Longrightarrow i < length\ Ps \Longrightarrow$
    $(Ps\ !\ i,\ s)\ {-c\rightarrow}\ (r,\ t) \Longrightarrow P) \Longrightarrow P$
  $\langle proof \rangle$

3.2.3 Computations

Sequential computations

type-synonym *'a confs = 'a conf list*

inductive-set *cptn* :: *'a confs set*
where
  CptnOne: $[(P,s)] \in cptn$
| *CptnEnv*: $(P,\ t)\#xs \in cptn \Longrightarrow (P,s)\#(P,t)\#xs \in cptn$
| *CptnComp*: $[\![(P,s)\ {-c\rightarrow}\ (Q,t);\ (Q,\ t)\#xs \in cptn\ ]\!] \Longrightarrow (P,s)\#(Q,t)\#xs \in cptn$

definition *cp* :: *('a com) option* ⇒ *'a* ⇒ *('a confs) set* **where**
  $cp\ P\ s \equiv \{l.\ l!0{=}(P,s) \wedge l \in cptn\}$

Parallel computations

type-synonym *'a par-confs = 'a par-conf list*

inductive-set *par-cptn* :: *'a par-confs set*
where
  ParCptnOne: $[(P,s)] \in$ *par-cptn*
| *ParCptnEnv*: $(P,t)\#xs \in$ *par-cptn* $\Longrightarrow (P,s)\#(P,t)\#xs \in$ *par-cptn*
| *ParCptnComp*: $[\![\ (P,s)\ {-pc\rightarrow}\ (Q,t);\ (Q,t)\#xs \in$ *par-cptn* $]\!] \Longrightarrow (P,s)\#(Q,t)\#xs$
$\in$ *par-cptn*

definition *par-cp* :: *'a par-com* ⇒ *'a* ⇒ *('a par-confs) set* **where**
  par-cp P s $\equiv \{l.\ l!0{=}(P,s) \wedge l \in$ *par-cptn*$\}$

3.2.4 Modular Definition of Computation

definition *lift* :: *'a com* ⇒ *'a conf* ⇒ *'a conf* **where**
  $lift\ Q \equiv \lambda(P,\ s).\ (if\ P{=}None\ then\ (Some\ Q,s)\ else\ (Some(Seq\ (the\ P)\ Q),\ s))$

inductive-set *cptn-mod* :: *('a confs) set*
where
  CptnModOne: $[(P,\ s)] \in$ *cptn-mod*
| *CptnModEnv*: $(P,\ t)\#xs \in$ *cptn-mod* $\Longrightarrow (P,\ s)\#(P,\ t)\#xs \in$ *cptn-mod*
| *CptnModNone*: $[\![(Some\ P,\ s)\ {-c\rightarrow}\ (None,\ t);\ (None,\ t)\#xs \in$ *cptn-mod* $]\!] \Longrightarrow$
$(Some\ P,s)\#(None,\ t)\#xs \in$*cptn-mod*

| *CptnModCondT*: ⟦(*Some P0*, *s*)#*ys* ∈ *cptn-mod*; *s* ∈ *b* ⟧ ⟹ (*Some*(*Cond b P0 P1*), *s*)#(*Some P0*, *s*)#*ys* ∈ *cptn-mod*
| *CptnModCondF*: ⟦(*Some P1*, *s*)#*ys* ∈ *cptn-mod*; *s* ∉ *b* ⟧ ⟹ (*Some*(*Cond b P0 P1*), *s*)#(*Some P1*, *s*)#*ys* ∈ *cptn-mod*
| *CptnModSeq1*: ⟦(*Some P0*, *s*)#*xs* ∈ *cptn-mod*; *zs*=*map* (*lift P1*) *xs* ⟧
⟹ (*Some*(*Seq P0 P1*), *s*)#*zs* ∈ *cptn-mod*
| *CptnModSeq2*:
⟦(*Some P0*, *s*)#*xs* ∈ *cptn-mod*; *fst*(*last* ((*Some P0*, *s*)#*xs*)) = *None*;
(*Some P1*, *snd*(*last* ((*Some P0*, *s*)#*xs*)))#*ys* ∈ *cptn-mod*;
zs=(*map* (*lift P1*) *xs*)@*ys* ⟧ ⟹ (*Some*(*Seq P0 P1*), *s*)#*zs* ∈ *cptn-mod*

| *CptnModWhile1*:
⟦ (*Some P*, *s*)#*xs* ∈ *cptn-mod*; *s* ∈ *b*; *zs*=*map* (*lift* (*While b P*)) *xs* ⟧
⟹ (*Some*(*While b P*), *s*)#(*Some*(*Seq P* (*While b P*)), *s*)#*zs* ∈ *cptn-mod*
| *CptnModWhile2*:
⟦ (*Some P*, *s*)#*xs* ∈ *cptn-mod*; *fst*(*last* ((*Some P*, *s*)#*xs*))=*None*; *s* ∈ *b*;
zs=(*map* (*lift* (*While b P*)) *xs*)@*ys*;
(*Some*(*While b P*), *snd*(*last* ((*Some P*, *s*)#*xs*)))#*ys* ∈ *cptn-mod*⟧
⟹ (*Some*(*While b P*), *s*)#(*Some*(*Seq P* (*While b P*)), *s*)#*zs* ∈ *cptn-mod*

3.2.5 Equivalence of Both Definitions.

lemma *last-length*: ((*a*#*xs*)!(*length xs*))=*last* (*a*#*xs*)
⟨*proof*⟩

lemma *div-seq* [*rule-format*]: *list* ∈ *cptn-mod* ⟹
(∀ *s P Q zs*. *list*=(*Some* (*Seq P Q*), *s*)#*zs* ⟶
(∃ *xs*. (*Some P*, *s*)#*xs* ∈ *cptn-mod* ∧ (*zs*=(*map* (*lift Q*) *xs*) ∨
(*fst*(((*Some P*, *s*)#*xs*)!*length xs*)=*None* ∧
(∃ *ys*. (*Some Q*, *snd*(((*Some P*, *s*)#*xs*)!*length xs*))#*ys* ∈ *cptn-mod*
∧ *zs*=(*map* (*lift* (*Q*)) *xs*)@*ys*)))))
⟨*proof*⟩

lemma *cptn-onlyif-cptn-mod-aux* [*rule-format*]:
∀ *s Q t xs*.((*Some a*, *s*), *Q*, *t*) ∈ *ctran* ⟶ (*Q*, *t*) # *xs* ∈ *cptn-mod*
⟶ (*Some a*, *s*) # (*Q*, *t*) # *xs* ∈ *cptn-mod*
⟨*proof*⟩

lemma *cptn-onlyif-cptn-mod* [*rule-format*]: *c* ∈ *cptn* ⟹ *c* ∈ *cptn-mod*
⟨*proof*⟩

lemma *lift-is-cptn*: *c*∈*cptn* ⟹ *map* (*lift P*) *c* ∈ *cptn*
⟨*proof*⟩

lemma *cptn-append-is-cptn* [*rule-format*]:
∀ *b a*. *b*#*c1*∈*cptn* ⟶ *a*#*c2*∈*cptn* ⟶ (*b*#*c1*)!*length c1*=*a* ⟶ *b*#*c1*@*c2*∈*cptn*
⟨*proof*⟩

lemma *last-lift*: ⟦*xs*≠[]; *fst*(*xs*!(*length xs* − (*Suc 0*)))=*None*⟧

$\Longrightarrow$ *fst((map (lift P) xs)!(length (map (lift P) xs)− (Suc 0)))=(Some P)*
⟨*proof*⟩

lemma *last-fst* [*rule-format*]: *P((a#x)!length x)* $\longrightarrow$ $\neg$*P a* $\longrightarrow$ *P (x!(length x* − *(Suc 0)))*
⟨*proof*⟩

lemma *last-fst-esp*:
fst(((Some a,s)#xs)!(length xs))=None $\Longrightarrow$ *fst(xs!(length xs* − *(Suc 0)))=None*
⟨*proof*⟩

lemma *last-snd*: *xs*≠[] $\Longrightarrow$
snd(((map (lift P) xs))!(length (map (lift P) xs) − (Suc 0)))=snd(xs!(length xs − (Suc 0)))
⟨*proof*⟩

lemma *Cons-lift*: *(Some (Seq P Q), s) # (map (lift Q) xs) = map (lift Q) ((Some P, s) # xs)*
⟨*proof*⟩

lemma *Cons-lift-append*:
(Some (Seq P Q), s) # (map (lift Q) xs) @ ys = map (lift Q) ((Some P, s) # xs)@ ys
⟨*proof*⟩

lemma *lift-nth*: *i<length xs* $\Longrightarrow$ *map (lift Q) xs ! i = lift Q (xs! i)*
⟨*proof*⟩

lemma *snd-lift*: *i< length xs* $\Longrightarrow$ *snd(lift Q (xs ! i))= snd (xs ! i)*
⟨*proof*⟩

lemma *cptn-if-cptn-mod*: *c* ∈ *cptn-mod* $\Longrightarrow$ *c* ∈ *cptn*
⟨*proof*⟩

theorem *cptn-iff-cptn-mod*: *(c* ∈ *cptn) = (c* ∈ *cptn-mod)*
⟨*proof*⟩

3.3 Validity of Correctness Formulas

3.3.1 Validity for Component Programs.

type-synonym *′a rgformula* =
′a com × *′a set* × *(′a* × *′a) set* × *(′a* × *′a) set* × *′a set*

definition *assum* :: *(′a set* × *(′a* × *′a) set)* ⇒ *(′a confs) set* **where**
assum ≡ λ*(pre, rely)*. {*c. snd(c!0)* ∈ *pre* ∧ (∀ *i. Suc i<length c* ⟶
c!i −e→ *c!(Suc i)* ⟶ *(snd(c!i), snd(c!Suc i))* ∈ *rely)*}

definition *comm* :: *((′a* × *′a) set* × *′a set)* ⇒ *(′a confs) set* **where**

$comm \equiv \lambda(guar, post).\ \{c.\ (\forall i.\ Suc\ i<length\ c \longrightarrow$
$c!i\ -c\rightarrow\ c!(Suc\ i) \longrightarrow (snd(c!i),\ snd(c!Suc\ i)) \in guar) \wedge$
$(fst\ (last\ c) = None \longrightarrow snd\ (last\ c) \in post)\}$

definition *com-validity* :: $'a\ com \Rightarrow 'a\ set \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set \Rightarrow 'a$ $set \Rightarrow bool$
$(\models$ - *sat* [-, -, -, -] [*60,0,0,0,0*] *45*$)$ **where**
$\models P\ sat\ [pre,\ rely,\ guar,\ post] \equiv$
$\forall s.\ cp\ (Some\ P)\ s \cap assum(pre,\ rely) \subseteq comm(guar,\ post)$

3.3.2 Validity for Parallel Programs.

definition *All-None* :: $(('a\ com)\ option)\ list \Rightarrow bool$ **where**
$All\text{-}None\ xs \equiv \forall c\in set\ xs.\ c=None$

definition *par-assum* :: $('a\ set \times ('a \times 'a)\ set) \Rightarrow ('a\ par\text{-}confs)\ set$ **where**
$par\text{-}assum \equiv \lambda(pre,\ rely).\ \{c.\ snd(c!0) \in pre \wedge (\forall i.\ Suc\ i<length\ c \longrightarrow$
$c!i\ -pe\rightarrow\ c!Suc\ i \longrightarrow (snd(c!i),\ snd(c!Suc\ i)) \in rely)\}$

definition *par-comm* :: $(('a \times 'a)\ set \times 'a\ set) \Rightarrow ('a\ par\text{-}confs)\ set$ **where**
$par\text{-}comm \equiv \lambda(guar,\ post).\ \{c.\ (\forall i.\ Suc\ i<length\ c \longrightarrow$
$c!i\ -pc\rightarrow\ c!Suc\ i \longrightarrow (snd(c!i),\ snd(c!Suc\ i)) \in guar) \wedge$
$(All\text{-}None\ (fst\ (last\ c)) \longrightarrow snd(\ last\ c) \in post)\}$

definition *par-com-validity* :: $'a\ \ par\text{-}com \Rightarrow 'a\ set \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)$ set
$\Rightarrow 'a\ set \Rightarrow bool$ $(\models$ - *SAT* [-, -, -, -] [*60,0,0,0,0*] *45*$)$ **where**
$\models Ps\ SAT\ [pre,\ rely,\ guar,\ post] \equiv$
$\forall s.\ par\text{-}cp\ Ps\ s \cap par\text{-}assum(pre,\ rely) \subseteq par\text{-}comm(guar,\ post)$

3.3.3 Compositionality of the Semantics

Definition of the conjoin operator

definition *same-length* :: $'a\ par\text{-}confs \Rightarrow ('a\ confs)\ list \Rightarrow bool$ **where**
$same\text{-}length\ c\ clist \equiv (\forall i<length\ clist.\ length(clist!i)=length\ c)$

definition *same-state* :: $'a\ par\text{-}confs \Rightarrow ('a\ confs)\ list \Rightarrow bool$ **where**
$same\text{-}state\ c\ clist \equiv (\forall i\ <length\ clist.\ \forall j<length\ c.\ snd(c!j) = snd((clist!i)!j))$

definition *same-program* :: $'a\ par\text{-}confs \Rightarrow ('a\ confs)\ list \Rightarrow bool$ **where**
$same\text{-}program\ c\ clist \equiv (\forall j<length\ c.\ fst(c!j) = map\ (\lambda x.\ fst(nth\ x\ j))\ clist)$

definition *compat-label* :: $'a\ par\text{-}confs \Rightarrow ('a\ confs)\ list \Rightarrow bool$ **where**
$compat\text{-}label\ c\ clist \equiv (\forall j.\ Suc\ j<length\ c \longrightarrow$
$(c!j\ -pc\rightarrow\ c!Suc\ j \wedge (\exists i<length\ clist.\ (clist!i)!j\ -c\rightarrow\ (clist!i)!\ Suc\ j \wedge$
$(\forall l<length\ clist.\ l\neq i \longrightarrow (clist!l)!j\ -e\rightarrow\ (clist!l)!\ Suc\ j))) \vee$
$(c!j\ -pe\rightarrow\ c!Suc\ j \wedge (\forall i<length\ clist.\ (clist!i)!j\ -e\rightarrow\ (clist!i)!\ Suc\ j)))$

definition $conjoin :: 'a\ par\text{-}confs \Rightarrow ('a\ confs)\ list \Rightarrow bool$ $(\text{-} \propto \text{-}\ [65,65]\ 64)$
where
$c \propto clist \equiv (same\text{-}length\ c\ clist) \wedge (same\text{-}state\ c\ clist) \wedge (same\text{-}program\ c\ clist) \wedge (compat\text{-}label\ c\ clist)$

Some previous lemmas

lemma *list-eq-if* [*rule-format*]:
$\forall ys.\ xs{=}ys \longrightarrow (length\ xs = length\ ys) \longrightarrow (\forall i{<}length\ xs.\ xs!i{=}ys!i)$
⟨*proof*⟩

lemma *list-eq*: $(length\ xs = length\ ys \wedge (\forall i{<}length\ xs.\ xs!i{=}ys!i)) = (xs{=}ys)$
⟨*proof*⟩

lemma *nth-tl*: $[\![\ ys!0{=}a;\ ys{\neq}[]\ ]\!] \Longrightarrow ys{=}(a\#(tl\ ys))$
⟨*proof*⟩

lemma *nth-tl-if* [*rule-format*]: $ys{\neq}[] \longrightarrow ys!0{=}a \longrightarrow P\ ys \longrightarrow P\ (a\#(tl\ ys))$
⟨*proof*⟩

lemma *nth-tl-onlyif* [*rule-format*]: $ys{\neq}[] \longrightarrow ys!0{=}a \longrightarrow P\ (a\#(tl\ ys)) \longrightarrow P\ ys$
⟨*proof*⟩

lemma *seq-not-eq1*: $Seq\ c1\ c2{\neq}c1$
⟨*proof*⟩

lemma *seq-not-eq2*: $Seq\ c1\ c2{\neq}c2$
⟨*proof*⟩

lemma *if-not-eq1*: $Cond\ b\ c1\ c2\ {\neq}c1$
⟨*proof*⟩

lemma *if-not-eq2*: $Cond\ b\ c1\ c2{\neq}c2$
⟨*proof*⟩

lemmas *seq-and-if-not-eq* [*simp*] = *seq-not-eq1 seq-not-eq2*
seq-not-eq1 [*THEN not-sym*] *seq-not-eq2* [*THEN not-sym*]
if-not-eq1 if-not-eq2 if-not-eq1 [*THEN not-sym*] *if-not-eq2* [*THEN not-sym*]

lemma *prog-not-eq-in-ctran-aux*:
assumes c: $(P,s)\ {-c\rightarrow}\ (Q,t)$
shows $P{\neq}Q$ ⟨*proof*⟩

lemma *prog-not-eq-in-ctran* [*simp*]: $\neg\ (P,s)\ {-c\rightarrow}\ (P,t)$
⟨*proof*⟩

lemma *prog-not-eq-in-par-ctran-aux* [*rule-format*]: $(P,s)\ {-pc\rightarrow}\ (Q,t) \Longrightarrow (P{\neq}Q)$
⟨*proof*⟩

lemma *prog-not-eq-in-par-ctran* [*simp*]: $\neg$ (*P*,*s*) $-pc\rightarrow$ (*P*,*t*)
⟨*proof*⟩

lemma *tl-in-cptn*: ⟦ *a*#*xs* ∈*cptn*; *xs*≠[] ⟧ ⟹ *xs*∈*cptn*
⟨*proof*⟩

lemma *tl-zero*[*rule-format*]:
P (*ys*!*Suc j*) ⟶ *Suc j*<*length ys* ⟶ *ys*≠[] ⟶ *P* (*tl*(*ys*)!*j*)
⟨*proof*⟩

3.3.4 The Semantics is Compositional

lemma *aux-if* [*rule-format*]:
∀ *xs s clist*. (*length clist* = *length xs* ∧ (∀ *i*<*length xs*. (*xs*!*i*,*s*)#*clist*!*i* ∈ *cptn*)
∧ ((*xs*, *s*)#*ys* ∝ *map* (λ*i*. (*fst i*,*s*)#*snd i*) (*zip xs clist*))
⟶ (*xs*, *s*)#*ys* ∈ *par-cptn*)
⟨*proof*⟩

lemma *aux-onlyif* [*rule-format*]: ∀ *xs s*. (*xs*, *s*)#*ys* ∈ *par-cptn* ⟶
(∃ *clist*. (*length clist* = *length xs*) ∧
(*xs*, *s*)#*ys* ∝ *map* (λ*i*. (*fst i*,*s*)#(*snd i*)) (*zip xs clist*) ∧
(∀ *i*<*length xs*. (*xs*!*i*,*s*)#(*clist*!*i*) ∈ *cptn*))
⟨*proof*⟩

lemma *one-iff-aux*: *xs*≠[] ⟹ (∀ *ys*. ((*xs*, *s*)#*ys* ∈ *par-cptn*) =
(∃ *clist*. *length clist*= *length xs* ∧
((*xs*, *s*)#*ys* ∝ *map* (λ*i*. (*fst i*,*s*)#(*snd i*)) (*zip xs clist*)) ∧
(∀ *i*<*length xs*. (*xs*!*i*,*s*)#(*clist*!*i*) ∈ *cptn*))) =
(*par-cp* (*xs*) *s* = {*c*. ∃ *clist*. (*length clist*)=(*length xs*) ∧
(∀ *i*<*length clist*. (*clist*!*i*) ∈ *cp*(*xs*!*i*) *s*) ∧ *c* ∝ *clist*})
⟨*proof*⟩

theorem *one*: *xs*≠[] ⟹
par-cp xs s = {*c*. ∃ *clist*. (*length clist*)=(*length xs*) ∧
(∀ *i*<*length clist*. (*clist*!*i*) ∈ *cp*(*xs*!*i*) *s*) ∧ *c* ∝ *clist*}
⟨*proof*⟩

end

3.4 The Proof System

theory *RG-Hoare* **imports** *RG-Tran* **begin**

3.4.1 Proof System for Component Programs

declare *Un-subset-iff* [*simp del*] *sup.bounded-iff* [*simp del*]

definition *stable* :: ′*a set* ⇒ (′*a* × ′*a*) *set* ⇒ *bool* **where**
stable ≡ λ*f g*. (∀ *x y*. *x* ∈ *f* ⟶ (*x*, *y*) ∈ *g* ⟶ *y* ∈ *f*)

inductive
rghoare :: $['a\ com,\ 'a\ set,\ ('a \times 'a)\ set,\ ('a \times 'a)\ set,\ 'a\ set] \Rightarrow bool$
($\vdash$ - *sat* [-, -, -, -] [*60*,*0*,*0*,*0*,*0*] *45*)
where
Basic: $[\![\ pre \subseteq \{s.\ f\ s \in post\};\ \{(s,t).\ s \in pre \wedge (t{=}f\ s \vee t{=}s)\} \subseteq guar;$
$stable\ pre\ rely;\ stable\ post\ rely\ ]\!]$
$\Longrightarrow \vdash Basic\ f\ sat\ [pre,\ rely,\ guar,\ post]$

| *Seq*: $[\![\vdash P\ sat\ [pre,\ rely,\ guar,\ mid]; \vdash Q\ sat\ [mid,\ rely,\ guar,\ post]\ ]\!]$
$\Longrightarrow \vdash Seq\ P\ Q\ sat\ [pre,\ rely,\ guar,\ post]$

| *Cond*: $[\![\ stable\ pre\ rely; \vdash P1\ sat\ [pre \cap b,\ rely,\ guar,\ post];$
$\vdash P2\ sat\ [pre \cap -b,\ rely,\ guar,\ post];\ \forall s.\ (s,s){\in}guar\ ]\!]$
$\Longrightarrow \vdash Cond\ b\ P1\ P2\ sat\ [pre,\ rely,\ guar,\ post]$

| *While*: $[\![\ stable\ pre\ rely;\ (pre \cap -b) \subseteq post;\ stable\ post\ rely;$
$\vdash P\ sat\ [pre \cap b,\ rely,\ guar,\ pre];\ \forall s.\ (s,s){\in}guar\ ]\!]$
$\Longrightarrow \vdash While\ b\ P\ sat\ [pre,\ rely,\ guar,\ post]$

| *Await*: $[\![\ stable\ pre\ rely;\ stable\ post\ rely;$
$\forall V. \vdash P\ sat\ [pre \cap b \cap \{V\},\ \{(s,\ t).\ s = t\},$
$UNIV,\ \{s.\ (V,\ s) \in guar\} \cap post]\ ]\!]$
$\Longrightarrow \vdash Await\ b\ P\ sat\ [pre,\ rely,\ guar,\ post]$

| *Conseq*: $[\![\ pre \subseteq pre';\ rely \subseteq rely';\ guar' \subseteq guar;\ post' \subseteq post;$
$\vdash P\ sat\ [pre',\ rely',\ guar',\ post']\ ]\!]$
$\Longrightarrow \vdash P\ sat\ [pre,\ rely,\ guar,\ post]$

definition *Pre* :: $'a\ rgformula \Rightarrow 'a\ set$ **where**
$Pre\ x \equiv fst(snd\ x)$

definition *Post* :: $'a\ rgformula \Rightarrow 'a\ set$ **where**
$Post\ x \equiv snd(snd(snd(snd\ x)))$

definition *Rely* :: $'a\ rgformula \Rightarrow ('a \times 'a)\ set$ **where**
$Rely\ x \equiv fst(snd(snd\ x))$

definition *Guar* :: $'a\ rgformula \Rightarrow ('a \times 'a)\ set$ **where**
$Guar\ x \equiv fst(snd(snd(snd\ x)))$

definition *Com* :: $'a\ rgformula \Rightarrow 'a\ com$ **where**
$Com\ x \equiv fst\ x$

3.4.2 Proof System for Parallel Programs

type-synonym *'a par-rgformula* =
$('a\ rgformula)\ list \times 'a\ set \times ('a \times 'a)\ set \times ('a \times 'a)\ set \times 'a\ set$

inductive
par-rghoare :: $('a\ rgformula)\ list \Rightarrow 'a\ set \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set \Rightarrow 'a\ set \Rightarrow bool$
$(\vdash\ \text{-}\ SAT\ [\text{-}, \text{-}, \text{-}, \text{-}]\ [60,0,0,0,0]\ 45)$
where
Parallel:
$[\![\ \forall i{<}length\ xs.\ rely \cup (\bigcup j{\in}\{j.\ j{<}length\ xs \wedge j{\neq}i\}.\ Guar(xs!j)) \subseteq Rely(xs!i);$
$(\bigcup j{\in}\{j.\ j{<}length\ xs\}.\ Guar(xs!j)) \subseteq guar;$
$pre \subseteq (\bigcap i{\in}\{i.\ i{<}length\ xs\}.\ Pre(xs!i));$
$(\bigcap i{\in}\{i.\ i{<}length\ xs\}.\ Post(xs!i)) \subseteq post;$
$\forall i{<}length\ xs.\ \vdash Com(xs!i)\ sat\ [Pre(xs!i),Rely(xs!i),Guar(xs!i),Post(xs!i)]\ ]\!]$
$\Longrightarrow\ \vdash xs\ SAT\ [pre, rely, guar, post]$

3.5 Soundness

Some previous lemmas

lemma *tl-of-assum-in-assum*:
$(P, s)\ \#\ (P, t)\ \#\ xs \in assum\ (pre, rely) \Longrightarrow stable\ pre\ rely$
$\Longrightarrow (P, t)\ \#\ xs \in assum\ (pre, rely)$
⟨*proof*⟩

lemma *etran-in-comm*:
$(P, t)\ \#\ xs \in comm(guar, post) \Longrightarrow (P, s)\ \#\ (P, t)\ \#\ xs \in comm(guar, post)$
⟨*proof*⟩

lemma *ctran-in-comm*:
$[\![(s, s) \in guar;\ (Q, s)\ \#\ xs \in comm(guar, post)]\!]$
$\Longrightarrow (P, s)\ \#\ (Q, s)\ \#\ xs \in comm(guar, post)$
⟨*proof*⟩

lemma *takecptn-is-cptn* [*rule-format*, *elim*!]:
$\forall j.\ c \in cptn \longrightarrow take\ (Suc\ j)\ c \in cptn$
⟨*proof*⟩

lemma *dropcptn-is-cptn* [*rule-format*,*elim*!]:
$\forall j{<}length\ c.\ c \in cptn \longrightarrow drop\ j\ c \in cptn$
⟨*proof*⟩

lemma *takepar-cptn-is-par-cptn* [*rule-format*,*elim*]:
$\forall j.\ c \in par\text{-}cptn \longrightarrow take\ (Suc\ j)\ c \in par\text{-}cptn$
⟨*proof*⟩

lemma *droppar-cptn-is-par-cptn* [*rule-format*]:
$\forall j{<}length\ c.\ c \in par\text{-}cptn \longrightarrow drop\ j\ c \in par\text{-}cptn$
⟨*proof*⟩

lemma *tl-of-cptn-is-cptn*: $[\![x\ \#\ xs \in cptn;\ xs \neq []]\!] \Longrightarrow xs\ \in cptn$
⟨*proof*⟩

lemma *not-ctran-None* [*rule-format*]:
$\forall s.\ (None,\ s)\#xs \in cptn \longrightarrow (\forall i<length\ xs.\ ((None,\ s)\#xs)!i\ -e\rightarrow\ xs!i)$
$\langle proof \rangle$

lemma *cptn-not-empty* [*simp*]:$[] \notin cptn$
$\langle proof \rangle$

lemma *etran-or-ctran* [*rule-format*]:
$\forall m\ i.\ x\in cptn \longrightarrow m \leq length\ x$
$\longrightarrow (\forall i.\ Suc\ i < m \longrightarrow \neg\ x!i\ -c\rightarrow\ x!Suc\ i) \longrightarrow Suc\ i < m$
$\longrightarrow x!i\ -e\rightarrow\ x!Suc\ i$
$\langle proof \rangle$

lemma *etran-or-ctran2* [*rule-format*]:
$\forall i.\ Suc\ i<length\ x \longrightarrow x\in cptn \longrightarrow (x!i\ -c\rightarrow\ x!Suc\ i \longrightarrow \neg\ x!i\ -e\rightarrow\ x!Suc\ i)$
$\vee\ (x!i\ -e\rightarrow\ x!Suc\ i \longrightarrow \neg\ x!i\ -c\rightarrow\ x!Suc\ i)$
$\langle proof \rangle$

lemma *etran-or-ctran2-disjI1*:
$[\![\ x\in cptn;\ Suc\ i<length\ x;\ x!i\ -c\rightarrow\ x!Suc\ i]\!] \Longrightarrow \neg\ x!i\ -e\rightarrow\ x!Suc\ i$
$\langle proof \rangle$

lemma *etran-or-ctran2-disjI2*:
$[\![\ x\in cptn;\ Suc\ i<length\ x;\ x!i\ -e\rightarrow\ x!Suc\ i]\!] \Longrightarrow \neg\ x!i\ -c\rightarrow\ x!Suc\ i$
$\langle proof \rangle$

lemma *not-ctran-None2* [*rule-format*]:
$[\![\ (None,\ s)\ \#\ xs\ \in cptn;\ i<length\ xs]\!] \Longrightarrow \neg\ ((None,\ s)\ \#\ xs)\ !\ i\ -c\rightarrow\ xs\ !\ i$
$\langle proof \rangle$

lemma *Ex-first-occurrence* [*rule-format*]: $P\ (n::nat) \longrightarrow (\exists m.\ P\ m \wedge (\forall i<m.\ \neg\ P\ i))$
$\langle proof \rangle$

lemma *stability* [*rule-format*]:
$\forall j\ k.\ x \in cptn \longrightarrow stable\ p\ rely \longrightarrow j\leq k \longrightarrow k<length\ x \longrightarrow snd(x!j)\in p \longrightarrow$
$(\forall i.\ (Suc\ i)<length\ x \longrightarrow$
$(x!i\ -e\rightarrow\ x!(Suc\ i)) \longrightarrow (snd(x!i),\ snd(x!(Suc\ i))) \in rely) \longrightarrow$
$(\forall i.\ j\leq i \wedge i<k \longrightarrow x!i\ -e\rightarrow\ x!Suc\ i) \longrightarrow snd(x!k)\in p \wedge fst(x!j)=fst(x!k)$
$\langle proof \rangle$

3.5.1 Soundness of the System for Component Programs

Soundness of the Basic rule

lemma *unique-ctran-Basic* [*rule-format*]:
$\forall s\ i.\ x \in cptn \longrightarrow x\ !\ 0 = (Some\ (Basic\ f),\ s) \longrightarrow$
$Suc\ i<length\ x \longrightarrow x!i\ -c\rightarrow\ x!Suc\ i \longrightarrow$
$(\forall j.\ Suc\ j<length\ x \longrightarrow i\neq j \longrightarrow x!j\ -e\rightarrow\ x!Suc\ j)$

$\langle proof \rangle$

lemma *exists-ctran-Basic-None* [*rule-format*]:
$\forall s\ i.\ x \in cptn \longrightarrow x\ !\ 0 = (Some\ (Basic\ f),\ s)$
$\longrightarrow i{<}length\ x \longrightarrow fst(x!i){=}None \longrightarrow (\exists j{<}i.\ x!j\ {-c\rightarrow}\ x!Suc\ j)$
$\langle proof \rangle$

lemma *Basic-sound*:
$[\![pre \subseteq \{s.\ f\ s \in post\};\ \{(s,\ t).\ s \in pre \wedge t = f\ s\} \subseteq guar;$
$stable\ pre\ rely;\ stable\ post\ rely]\!]$
$\Longrightarrow \models Basic\ f\ sat\ [pre,\ rely,\ guar,\ post]$
$\langle proof \rangle$

Soundness of the Await rule

lemma *unique-ctran-Await* [*rule-format*]:
$\forall s\ i.\ x \in cptn \longrightarrow x\ !\ 0 = (Some\ (Await\ b\ c),\ s) \longrightarrow$
$Suc\ i{<}length\ x \longrightarrow x!i\ {-c\rightarrow}\ x!Suc\ i \longrightarrow$
$(\forall j.\ Suc\ j{<}length\ x \longrightarrow i{\neq}j \longrightarrow x!j\ {-e\rightarrow}\ x!Suc\ j)$
$\langle proof \rangle$

lemma *exists-ctran-Await-None* [*rule-format*]:
$\forall s\ i.\ x \in cptn \longrightarrow x\ !\ 0 = (Some\ (Await\ b\ c),\ s)$
$\longrightarrow i{<}length\ x \longrightarrow fst(x!i){=}None \longrightarrow (\exists j{<}i.\ x!j\ {-c\rightarrow}\ x!Suc\ j)$
$\langle proof \rangle$

lemma *Star-imp-cptn*:
$(P,\ s)\ {-c{*}\rightarrow}\ (R,\ t) \Longrightarrow \exists l \in cp\ P\ s.\ (last\ l){=}(R,\ t)$
$\wedge (\forall i.\ Suc\ i{<}length\ l \longrightarrow l!i\ {-c\rightarrow}\ l!Suc\ i)$
$\langle proof \rangle$

lemma *Await-sound*:
$[\![stable\ pre\ rely;\ stable\ post\ rely;$
$\forall V.\ \vdash P\ sat\ [pre \cap b \cap \{s.\ s = V\},\ \{(s,\ t).\ s = t\},$
$UNIV,\ \{s.\ (V,\ s) \in guar\} \cap post] \wedge$
$\models P\ sat\ [pre \cap b \cap \{s.\ s = V\},\ \{(s,\ t).\ s = t\},$
$UNIV,\ \{s.\ (V,\ s) \in guar\} \cap post]\ ]\!]$
$\Longrightarrow \models Await\ b\ P\ sat\ [pre,\ rely,\ guar,\ post]$
$\langle proof \rangle$

Soundness of the Conditional rule

lemma *Cond-sound*:
$[\![\ stable\ pre\ rely;\ \models P1\ sat\ [pre \cap b,\ rely,\ guar,\ post];$
$\models P2\ sat\ [pre \cap -\ b,\ rely,\ guar,\ post];\ \forall s.\ (s,s){\in}guar]\!]$
$\Longrightarrow \models (Cond\ b\ P1\ P2)\ sat\ [pre,\ rely,\ guar,\ post]$
$\langle proof \rangle$

Soundness of the Sequential rule

inductive-cases *Seq-cases* [*elim*!]: (*Some* (*Seq P Q*), *s*) $-c\rightarrow$ *t*

lemma *last-lift-not-None*: $fst\ ((lift\ Q)\ ((x\#xs)!(length\ xs))) \neq None$
⟨*proof*⟩

lemma *Seq-sound1* [*rule-format*]:
$x\in\ cptn\text{-}mod \Longrightarrow \forall s\ P.\ x\ !0=(Some\ (Seq\ P\ Q),\ s) \longrightarrow$
$(\forall i<length\ x.\ fst(x!i)\neq Some\ Q) \longrightarrow$
$(\exists xs\in\ cp\ (Some\ P)\ s.\ x=map\ (lift\ Q)\ xs)$
⟨*proof*⟩

lemma *Seq-sound2* [*rule-format*]:
$x \in cptn \Longrightarrow \forall s\ P\ i.\ x!0=(Some\ (Seq\ P\ Q),\ s) \longrightarrow i<length\ x$
$\longrightarrow fst(x!i)=Some\ Q \longrightarrow$
$(\forall j<i.\ fst(x!j)\neq(Some\ Q)) \longrightarrow$
$(\exists xs\ ys.\ xs \in cp\ (Some\ P)\ s \wedge length\ xs=Suc\ i$
$\wedge\ ys \in cp\ (Some\ Q)\ (snd(xs\ !i)) \wedge x=(map\ (lift\ Q)\ xs)@tl\ ys)$
⟨*proof*⟩

lemma *last-lift-not-None2*: $fst\ ((lift\ Q)\ (last\ (x\#xs))) \neq None$
⟨*proof*⟩

lemma *Seq-sound*:
$[\![\models P\ sat\ [pre,\ rely,\ guar,\ mid];\ \models Q\ sat\ [mid,\ rely,\ guar,\ post]]\!]$
$\Longrightarrow\ \models Seq\ P\ Q\ sat\ [pre,\ rely,\ guar,\ post]$
⟨*proof*⟩

Soundness of the While rule

lemma *last-append*[*rule-format*]:
$\forall xs.\ ys\neq[] \longrightarrow ((xs@ys)!(length\ (xs@ys) - (Suc\ 0)))=(ys!(length\ ys - (Suc\ 0)))$
⟨*proof*⟩

lemma *assum-after-body*:
$[\![\ \models P\ sat\ [pre \cap b,\ rely,\ guar,\ pre];$
$(Some\ P,\ s)\ \#\ xs \in cptn\text{-}mod;\ fst\ (last\ ((Some\ P,\ s)\ \#\ xs)) = None;\ s \in b;$
$(Some\ (While\ b\ P),\ s)\ \#\ (Some\ (Seq\ P\ (While\ b\ P)),\ s)\ \#$
$map\ (lift\ (While\ b\ P))\ xs\ @\ ys \in assum\ (pre,\ rely)]\!]$
$\Longrightarrow (Some\ (While\ b\ P),\ snd\ (last\ ((Some\ P,\ s)\ \#\ xs)))\ \#\ ys \in assum\ (pre,\ rely)$
⟨*proof*⟩

lemma *While-sound-aux* [*rule-format*]:
$[\![\ pre \cap -\ b \subseteq post;\ \models P\ sat\ [pre \cap b,\ rely,\ guar,\ pre];\ \forall s.\ (s,\ s) \in guar;$
$stable\ pre\ rely;\ \ stable\ post\ rely;\ x \in cptn\text{-}mod\ ]\!]$
$\Longrightarrow\ \ \forall s\ xs.\ x=(Some(While\ b\ P),s)\#xs \longrightarrow x\in assum(pre,\ rely) \longrightarrow x \in comm$
$(guar,\ post)$
⟨*proof*⟩

lemma *While-sound*:
$[\![$*stable pre rely*; *pre* $\cap -$ *b* $\subseteq$ *post*; *stable post rely*;
$\models$ *P sat* [*pre* $\cap$ *b*, *rely*, *guar*, *pre*]; $\forall$ *s*. (*s*,*s*)$\in$*guar*$]\!]$
$\Longrightarrow$ $\models$ *While b P sat* [*pre*, *rely*, *guar*, *post*]
$\langle$*proof*$\rangle$

Soundness of the Rule of Consequence

lemma *Conseq-sound*:
$[\![$*pre* $\subseteq$ *pre'*; *rely* $\subseteq$ *rely'*; *guar'* $\subseteq$ *guar*; *post'* $\subseteq$ *post*;
$\models$ *P sat* [*pre'*, *rely'*, *guar'*, *post'*]$]\!]$
$\Longrightarrow$ $\models$ *P sat* [*pre*, *rely*, *guar*, *post*]
$\langle$*proof*$\rangle$

Soundness of the system for sequential component programs

theorem *rgsound*:
$\vdash$ *P sat* [*pre*, *rely*, *guar*, *post*] $\Longrightarrow$ $\models$ *P sat* [*pre*, *rely*, *guar*, *post*]
$\langle$*proof*$\rangle$

3.5.2 Soundness of the System for Parallel Programs

definition *ParallelCom* :: (*'a rgformula*) *list* $\Rightarrow$ *'a par-com* **where**
ParallelCom Ps $\equiv$ *map* (*Some* $\circ$ *fst*) *Ps*

lemma *two*:
$[\![$ $\forall$ *i*<*length xs*. *rely* $\cup$ ($\bigcup$*j*$\in${*j*. *j* < *length xs* $\wedge$ *j* $\neq$ *i*}. *Guar* (*xs* ! *j*))
$\subseteq$ *Rely* (*xs* ! *i*);
pre $\subseteq$ ($\bigcap$*i*$\in${*i*. *i* < *length xs*}. *Pre* (*xs* ! *i*));
$\forall$ *i*<*length xs*.
$\models$ *Com* (*xs* ! *i*) *sat* [*Pre* (*xs* ! *i*), *Rely* (*xs* ! *i*), *Guar* (*xs* ! *i*), *Post* (*xs* ! *i*)];
length xs=*length clist*; *x* $\in$ *par-cp* (*ParallelCom xs*) *s*; *x*$\in$*par-assum*(*pre*, *rely*);
$\forall$ *i*<*length clist*. *clist*!*i*$\in$*cp* (*Some*(*Com*(*xs*!*i*))) *s*; *x* $\propto$ *clist* $]\!]$
$\Longrightarrow$ $\forall$ *j i*. *i*<*length clist* $\wedge$ *Suc j*<*length x* $\longrightarrow$ (*clist*!*i*!*j*) $-c\rightarrow$ (*clist*!*i*!*Suc j*)
$\longrightarrow$ (*snd*(*clist*!*i*!*j*), *snd*(*clist*!*i*!*Suc j*)) $\in$ *Guar*(*xs*!*i*)
$\langle$*proof*$\rangle$

lemma *three* [*rule-format*]:
$[\![$ *xs*$\neq$[]; $\forall$ *i*<*length xs*. *rely* $\cup$ ($\bigcup$*j*$\in${*j*. *j* < *length xs* $\wedge$ *j* $\neq$ *i*}. *Guar* (*xs* ! *j*))
$\subseteq$ *Rely* (*xs* ! *i*);
pre $\subseteq$ ($\bigcap$*i*$\in${*i*. *i* < *length xs*}. *Pre* (*xs* ! *i*));
$\forall$ *i*<*length xs*.
$\models$ *Com* (*xs* ! *i*) *sat* [*Pre* (*xs* ! *i*), *Rely* (*xs* ! *i*), *Guar* (*xs* ! *i*), *Post* (*xs* ! *i*)];
length xs=*length clist*; *x* $\in$ *par-cp* (*ParallelCom xs*) *s*; *x* $\in$ *par-assum*(*pre*, *rely*);
$\forall$ *i*<*length clist*. *clist*!*i*$\in$*cp* (*Some*(*Com*(*xs*!*i*))) *s*; *x* $\propto$ *clist* $]\!]$
$\Longrightarrow$ $\forall$ *j i*. *i*<*length clist* $\wedge$ *Suc j*<*length x* $\longrightarrow$ (*clist*!*i*!*j*) $-e\rightarrow$ (*clist*!*i*!*Suc j*)
$\longrightarrow$ (*snd*(*clist*!*i*!*j*), *snd*(*clist*!*i*!*Suc j*)) $\in$ *rely* $\cup$ ($\bigcup$*j*$\in${*j*. *j* < *length xs* $\wedge$ *j* $\neq$ *i*}.
Guar (*xs* ! *j*))

$\langle proof \rangle$

lemma *four*:
$[\![xs{\neq}[]; \forall\, i < length\ xs.\ rely \cup (\bigcup j{\in}\{j.\ j < length\ xs \wedge j \neq i\}.\ Guar\ (xs\ !\ j))$
$\subseteq Rely\ (xs\ !\ i);$
$(\bigcup j{\in}\{j.\ j < length\ xs\}.\ Guar\ (xs\ !\ j)) \subseteq guar;$
$pre \subseteq (\bigcap i{\in}\{i.\ i < length\ xs\}.\ Pre\ (xs\ !\ i));$
$\forall\, i < length\ xs.$
$\models Com\ (xs\ !\ i)\ sat\ [Pre\ (xs\ !\ i),\ Rely\ (xs\ !\ i),\ Guar\ (xs\ !\ i),\ Post\ (xs\ !\ i)];$
$x \in par\text{-}cp\ (ParallelCom\ xs)\ s;\ x \in par\text{-}assum\ (pre,\ rely);\ Suc\ i < length\ x;$
$x\ !\ i\ {-pc\rightarrow}\ x\ !\ Suc\ i]\!]$
$\Longrightarrow (snd\ (x\ !\ i),\ snd\ (x\ !\ Suc\ i)) \in guar$
$\langle proof \rangle$

lemma *parcptn-not-empty* [*simp*]:$[] \notin par\text{-}cptn$
$\langle proof \rangle$

lemma *five*:
$[\![xs{\neq}[]; \forall\, i{<}length\ xs.\ rely \cup (\bigcup j{\in}\{j.\ j < length\ xs \wedge j \neq i\}.\ Guar\ (xs\ !\ j))$
$\subseteq Rely\ (xs\ !\ i);$
$pre \subseteq (\bigcap i{\in}\{i.\ i < length\ xs\}.\ Pre\ (xs\ !\ i));$
$(\bigcap i{\in}\{i.\ i < length\ xs\}.\ Post\ (xs\ !\ i)) \subseteq post;$
$\forall\, i < length\ xs.$
$\models Com\ (xs\ !\ i)\ sat\ [Pre\ (xs\ !\ i),\ Rely\ (xs\ !\ i),\ Guar\ (xs\ !\ i),\ Post\ (xs\ !\ i)];$
$x \in par\text{-}cp\ (ParallelCom\ xs)\ s;\ x \in par\text{-}assum\ (pre,\ rely);$
$All\text{-}None\ (fst\ (last\ x))\ ]\!] \Longrightarrow snd\ (last\ x) \in post$
$\langle proof \rangle$

lemma *ParallelEmpty* [*rule-format*]:
$\forall\, i\ s.\ x \in par\text{-}cp\ (ParallelCom\ [])\ s \longrightarrow$
$Suc\ i < length\ x \longrightarrow (x\ !\ i,\ x\ !\ Suc\ i) \notin par\text{-}ctran$
$\langle proof \rangle$

theorem *par-rgsound*:
$\vdash c\ SAT\ [pre,\ rely,\ guar,\ post] \Longrightarrow$
$\models (ParallelCom\ c)\ SAT\ [pre,\ rely,\ guar,\ post]$
$\langle proof \rangle$

end

3.6 Concrete Syntax

theory *RG-Syntax*
imports *RG-Hoare Quote-Antiquote*
begin

abbreviation *Skip* :: *'a com* (*SKIP*)
where $SKIP \equiv Basic\ id$

notation *Seq* ((-;;/ -) [60,61] 60)

syntax
  -*Assign* :: *idt* ⇒ *'b* ⇒ *'a com* ((´- :=/ -) [70, 65] 61)
  -*Cond* :: *'a bexp* ⇒ *'a com* ⇒ *'a com* ⇒ *'a com* ((0IF -/ THEN -/ ELSE -/FI) [0, 0, 0] 61)
  -*Cond2* :: *'a bexp* ⇒ *'a com* ⇒ *'a com* ((0IF - THEN - FI) [0,0] 56)
  -*While* :: *'a bexp* ⇒ *'a com* ⇒ *'a com* ((0WHILE - /DO - /OD) [0, 0] 61)
  -*Await* :: *'a bexp* ⇒ *'a com* ⇒ *'a com* ((0AWAIT - /THEN /- /END) [0,0] 61)
  -*Atom* :: *'a com* ⇒ *'a com* ((⟨-⟩) 61)
  -*Wait* :: *'a bexp* ⇒ *'a com* ((0WAIT - END) 61)

translations
  ´x := a ⇀ *CONST Basic* ≪´(-*update-name x* (λ-. *a*))≫
  IF b THEN c1 ELSE c2 FI ⇀ *CONST Cond* {|*b*|} *c1 c2*
  IF b THEN c FI ⇌ *IF b THEN c ELSE SKIP FI*
  WHILE b DO c OD ⇀ *CONST While* {|*b*|} *c*
  AWAIT b THEN c END ⇌ *CONST Await* {|*b*|} *c*
  ⟨*c*⟩ ⇌ *AWAIT CONST True THEN c END*
  WAIT b END ⇌ *AWAIT b THEN SKIP END*

nonterminal *prgs*

syntax
  -*PAR* :: *prgs* ⇒ *'a* (*COBEGIN*//-//*COEND* 60)
  -*prg* :: *'a* ⇒ *prgs* (- 57)
  -*prgs* :: [*'a*, *prgs*] ⇒ *prgs* (-//‖//- [60,57] 57)

translations
  -*prg a* ⇀ [*a*]
  -*prgs a ps* ⇀ *a* # *ps*
  -*PAR ps* ⇀ *ps*

syntax
  -*prg-scheme* :: [*'a*, *'a*, *'a*, *'a*] ⇒ *prgs* (*SCHEME* [- ≤ - < -] - [0,0,0,60] 57)

translations
  -*prg-scheme j i k c* ⇌ (*CONST map* (λ*i*. *c*) [*j*..<*k*])

Translations for variables before and after a transition:

syntax
  -*before* :: *id* ⇒ *'a* (ᵒ-)
  -*after* :: *id* ⇒ *'a* (ᵃ-)

translations
  ᵒ*x* ⇌ *x* ´*CONST fst*
  ᵃ*x* ⇌ *x* ´*CONST snd*

⟨*ML*⟩

end

3.7 Examples

theory *RG-Examples*
imports *RG-Syntax*
begin

lemmas *definitions* [*simp*]= *stable-def Pre-def Rely-def Guar-def Post-def Com-def*

3.7.1 Set Elements of an Array to Zero

lemma *le-less-trans2*: $[\![(j::nat)<k;\ i\leq j]\!] \Longrightarrow i<k$
⟨*proof*⟩

lemma *add-le-less-mono*: $[\![\ (a::nat) < c;\ b\leq d\ ]\!] \Longrightarrow a + b < c + d$
⟨*proof*⟩

record *Example1* =
A :: *nat list*

lemma *Example1*:
$\vdash COBEGIN$
$\quad SCHEME\ [0 \leq i < n]$
$\quad (\acute{}A := \acute{}A\ [i := 0],$
$\quad \{\!|\ n < length\ \acute{}A\ |\!\},$
$\quad \{\!|\ length\ {}^{o}A = length\ {}^{a}A \wedge {}^{o}A\ !\ i = {}^{a}A\ !\ i\ |\!\},$
$\quad \{\!|\ length\ {}^{o}A = length\ {}^{a}A \wedge (\forall j<n.\ i \neq j \longrightarrow {}^{o}A\ !\ j = {}^{a}A\ !\ j)\ |\!\},$
$\quad \{\!|\ \acute{}A\ !\ i = 0\ |\!\})$
$\quad COEND$
$SAT\ [\{\!|\ n < length\ \acute{}A\ |\!\},\ \{\!|\ {}^{o}A = {}^{a}A\ |\!\},\ \{\!|\ True\ |\!\},\ \{\!|\ \forall i < n.\ \acute{}A\ !\ i = 0\ |\!\}]$
⟨*proof*⟩

lemma *Example1-parameterized*:
$k < t \Longrightarrow$
$\vdash COBEGIN$
$\quad SCHEME\ [k{*}n\leq i<(Suc\ k){*}n]\ (\acute{}A:=\acute{}A[i:=0],$
$\quad \{\!|t{*}n < length\ \acute{}A|\!\},$
$\quad \{\!|t{*}n < length\ {}^{o}A \wedge length\ {}^{o}A=length\ {}^{a}A \wedge {}^{o}A!i = {}^{a}A!i|\!\},$
$\quad \{\!|t{*}n < length\ {}^{o}A \wedge length\ {}^{o}A=length\ {}^{a}A \wedge (\forall j<length\ {}^{o}A\ .\ i\neq j \longrightarrow {}^{o}A!j =$
${}^{a}A!j)|\!\},$
$\quad \{\!|\acute{}A!i=0|\!\})$
$\quad COEND$
$SAT\ [\{\!|t{*}n < length\ \acute{}A|\!\},$
$\quad \{\!|t{*}n < length\ {}^{o}A \wedge length\ {}^{o}A=length\ {}^{a}A \wedge (\forall i<n.\ {}^{o}A!(k{*}n+i)={}^{a}A!(k{*}n+i))|\!\},$
$\quad \{\!|t{*}n < length\ {}^{o}A \wedge length\ {}^{o}A=length\ {}^{a}A \wedge$

$(\forall\, i{<}length\ ^{\mathrm{o}}A\ .\ (i{<}k{*}n \longrightarrow {}^{\mathrm{o}}A!i = {}^{\mathrm{a}}A!i) \wedge ((Suc\ k){*}n \le i \longrightarrow {}^{\mathrm{o}}A!i =$
${}^{\mathrm{a}}A!i))\}$,
$\{\forall\, i{<}n.\ {}'A!(k{*}n{+}i) = 0\}]$
$\langle proof \rangle$

3.7.2 Increment a Variable in Parallel

Two components

record *Example2* =
x :: *nat*
$c\text{-}0$ :: *nat*
$c\text{-}1$ :: *nat*

lemma *Example2*:
$\vdash$ *COBEGIN*
$(\langle\ {}'x{:=}{}'x{+}1;;\ {}'c\text{-}0{:=}{}'c\text{-}0 + 1\ \rangle,$
$\{{}'x{=}{}'c\text{-}0 + {}'c\text{-}1\ \wedge\ {}'c\text{-}0{=}0\},$
$\{{}^{\mathrm{o}}c\text{-}0 = {}^{\mathrm{a}}c\text{-}0\ \wedge$
$({}^{\mathrm{o}}x{=}{}^{\mathrm{o}}c\text{-}0 + {}^{\mathrm{o}}c\text{-}1$
$\longrightarrow {}^{\mathrm{a}}x = {}^{\mathrm{a}}c\text{-}0 + {}^{\mathrm{a}}c\text{-}1)\},$
$\{{}^{\mathrm{o}}c\text{-}1 = {}^{\mathrm{a}}c\text{-}1\ \wedge$
$({}^{\mathrm{o}}x{=}{}^{\mathrm{o}}c\text{-}0 + {}^{\mathrm{o}}c\text{-}1$
$\longrightarrow {}^{\mathrm{a}}x = {}^{\mathrm{a}}c\text{-}0 + {}^{\mathrm{a}}c\text{-}1)\},$
$\{{}'x{=}{}'c\text{-}0 + {}'c\text{-}1\ \wedge\ {}'c\text{-}0{=}1\ \})$
$\|$
$(\langle\ {}'x{:=}{}'x{+}1;;\ {}'c\text{-}1{:=}{}'c\text{-}1{+}1\ \rangle,$
$\{{}'x{=}{}'c\text{-}0 + {}'c\text{-}1\ \wedge\ {}'c\text{-}1{=}0\ \},$
$\{{}^{\mathrm{o}}c\text{-}1 = {}^{\mathrm{a}}c\text{-}1\ \wedge$
$({}^{\mathrm{o}}x{=}{}^{\mathrm{o}}c\text{-}0 + {}^{\mathrm{o}}c\text{-}1$
$\longrightarrow {}^{\mathrm{a}}x = {}^{\mathrm{a}}c\text{-}0 + {}^{\mathrm{a}}c\text{-}1)\},$
$\{{}^{\mathrm{o}}c\text{-}0 = {}^{\mathrm{a}}c\text{-}0\ \wedge$
$({}^{\mathrm{o}}x{=}{}^{\mathrm{o}}c\text{-}0 + {}^{\mathrm{o}}c\text{-}1$
$\longrightarrow {}^{\mathrm{a}}x = {}^{\mathrm{a}}c\text{-}0 + {}^{\mathrm{a}}c\text{-}1)\},$
$\{{}'x{=}{}'c\text{-}0 + {}'c\text{-}1\ \wedge\ {}'c\text{-}1{=}1\})$
COEND
$SAT\ [\{{}'x{=}0\ \wedge\ {}'c\text{-}0{=}0\ \wedge\ {}'c\text{-}1{=}0\},$
$\{{}^{\mathrm{o}}x{=}{}^{\mathrm{a}}x\ \wedge\ {}^{\mathrm{o}}c\text{-}0{=}\ {}^{\mathrm{a}}c\text{-}0\ \wedge\ {}^{\mathrm{o}}c\text{-}1{=}{}^{\mathrm{a}}c\text{-}1\},$
$\{True\},$
$\{{}'x{=}2\}]$
$\langle proof \rangle$

Parameterized

lemma *Example2-lemma2-aux*: $j{<}n \Longrightarrow$
$(\sum i{=}0..{<}n.\ (b\ i{::}nat)) =$
$(\sum i{=}0..{<}j.\ b\ i) + b\ j + (\sum i{=}0..{<}n{-}(Suc\ j)\ .\ b\ (Suc\ j + i))$
$\langle proof \rangle$

lemma *Example2-lemma2-aux2*:

$j \le s \Longrightarrow (\sum i::nat{=}0..{<}j.\ (b\ (s{:=}t))\ i) = (\sum i{=}0..{<}j.\ b\ i)$
⟨*proof*⟩

lemma *Example2-lemma2*:
$[\![j{<}n;\ b\ j{=}0]\!] \Longrightarrow Suc\ (\sum i::nat{=}0..{<}n.\ b\ i){=}(\sum i{=}0..{<}n.\ (b\ (j := Suc\ 0))\ i)$
⟨*proof*⟩

lemma *Example2-lemma2-Suc0*: $[\![j{<}n;\ b\ j{=}0]\!] \Longrightarrow$
$Suc\ (\sum i::nat{=}0..{<}\ n.\ b\ i){=}(\sum i{=}0..{<}\ n.\ (b\ (j{:=}Suc\ 0))\ i)$
⟨*proof*⟩

record *Example2-parameterized* =
$C :: nat \Rightarrow nat$
$y :: nat$

lemma *Example2-parameterized*: $0{<}n \Longrightarrow$
$\vdash COBEGIN\ SCHEME\ \ [0{\le}i{<}n]$
$(\langle\ {}^\acute{}y{:=}{}^\acute{}y{+}1;;\ {}^\acute{}C{:=}{}^\acute{}C\ (i{:=}1)\ \rangle,$
$\{\!|{}^\acute{}y{=}(\sum i{=}0..{<}n.\ {}^\acute{}C\ i) \wedge {}^\acute{}C\ i{=}0|\!\},$
$\{\!|{}^{o}C\ i = {}^{a}C\ i \wedge$
$({}^{o}y{=}(\sum i{=}0..{<}n.\ {}^{o}C\ i) \longrightarrow {}^{a}y = (\sum i{=}0..{<}n.\ {}^{a}C\ i))|\!\},$
$\{\!|(\forall j{<}n.\ i{\ne}j \longrightarrow {}^{o}C\ j = {}^{a}C\ j) \wedge$
$({}^{o}y{=}(\sum i{=}0..{<}n.\ {}^{o}C\ i) \longrightarrow {}^{a}y = (\sum i{=}0..{<}n.\ {}^{a}C\ i))|\!\},$
$\{\!|{}^\acute{}y{=}(\sum i{=}0..{<}n.\ {}^\acute{}C\ i) \wedge {}^\acute{}C\ i{=}1|\!\})$
$COEND$
$SAT\ [\{\!|{}^\acute{}y{=}0 \wedge (\sum i{=}0..{<}n.\ {}^\acute{}C\ i){=}0\ |\!\},\ \{\!|{}^{o}C{=}{}^{a}C \wedge {}^{o}y{=}{}^{a}y|\!\},\ \{\!|True|\!\},\ \{\!|{}^\acute{}y{=}n|\!\}]$
⟨*proof*⟩

3.7.3 Find Least Element

A previous lemma:

lemma *mod-aux* :$[\![i < (n::nat);\ a\ mod\ n = i;\ \ j < a + n;\ j\ mod\ n = i;\ a < j]\!]$
$\Longrightarrow False$
⟨*proof*⟩

record *Example3* =
$X :: nat \Rightarrow nat$
$Y :: nat \Rightarrow nat$

lemma *Example3*: $m\ mod\ n{=}0 \Longrightarrow$
$\vdash COBEGIN$
$SCHEME\ [0{\le}i{<}n]$
$(WHILE\ (\forall j{<}n.\ {}^\acute{}X\ i < {}^\acute{}Y\ j)\ \ DO$
$IF\ P(B!({}^\acute{}X\ i))\ THEN\ {}^\acute{}Y{:=}{}^\acute{}Y\ (i{:=}{}^\acute{}X\ i)$
$ELSE\ {}^\acute{}X{:=}\ {}^\acute{}X\ (i{:=}({}^\acute{}X\ i){+}\ n)\ FI$
$OD,$
$\{\!|({}^\acute{}X\ i)\ mod\ n{=}i \wedge (\forall j{<}{}^\acute{}X\ i.\ j\ mod\ n{=}i \longrightarrow \neg P(B!j)) \wedge ({}^\acute{}Y\ i{<}m \longrightarrow P(B!({}^\acute{}Y$
$i)) \wedge {}^\acute{}Y\ i{\le}\ m{+}i)|\!\},$
$\{\!|(\forall j{<}n.\ i{\ne}j \longrightarrow {}^{a}Y\ j \le {}^{o}Y\ j) \wedge {}^{o}X\ i = {}^{a}X\ i \wedge$

$^{o}Y\ i = {}^{a}Y\ i$⦄,
⦃$(\forall j<n.\ i\neq j \longrightarrow {}^{o}X\ j = {}^{a}X\ j \wedge {}^{o}Y\ j = {}^{a}Y\ j) \wedge$
$^{a}Y\ i \leq {}^{o}Y\ i$⦄,
⦃$(\acute{}X\ i)\ mod\ n{=}i \wedge (\forall j<\acute{}X\ i.\ j\ mod\ n{=}i \longrightarrow \neg P(B!j)) \wedge (\acute{}Y\ i<m \longrightarrow P(B!(\acute{}Y\ i)) \wedge \acute{}Y\ i\leq m{+}i) \wedge (\exists j<n.\ \acute{}Y\ j \leq \acute{}X\ i)$ ⦄)
COEND
SAT [⦃ $\forall i<n.\ \acute{}X\ i{=}i \wedge \acute{}Y\ i{=}m{+}i$ ⦄,⦃$^{o}X{=}{}^{a}X \wedge {}^{o}Y{=}{}^{a}Y$⦄,⦃*True*⦄,
⦃$\forall i<n.\ (\acute{}X\ i)\ mod\ n{=}i \wedge (\forall j<\acute{}X\ i.\ j\ mod\ n{=}i \longrightarrow \neg P(B!j)) \wedge$
$(\acute{}Y\ i<m \longrightarrow P(B!(\acute{}Y\ i)) \wedge \acute{}Y\ i\leq m{+}i) \wedge (\exists j<n.\ \acute{}Y\ j \leq \acute{}X\ i)$⦄]
⟨*proof*⟩

Same but with a list as auxiliary variable:

record *Example3-list* =
X :: *nat list*
Y :: *nat list*

lemma *Example3-list*: $m\ mod\ n{=}0 \Longrightarrow \vdash$ (*COBEGIN SCHEME* [$0\leq i<n$]
(*WHILE* $(\forall j<n.\ \acute{}X!i < \acute{}Y!j)$ *DO*
IF $P(B!(\acute{}X!i))$ *THEN* $\acute{}Y{:=}\acute{}Y[i{:=}\acute{}X!i]$ *ELSE* $\acute{}X{:=}\ \acute{}X[i{:=}(\acute{}X!i){+}\ n]$ *FI*
OD,
⦃$n<length\ \acute{}X \wedge n<length\ \acute{}Y \wedge (\acute{}X!i)\ mod\ n{=}i \wedge (\forall j<\acute{}X!i.\ j\ mod\ n{=}i \longrightarrow \neg P(B!j)) \wedge (\acute{}Y!i<m \longrightarrow P(B!(\acute{}Y!i)) \wedge \acute{}Y!i\leq m{+}i)$⦄,
⦃$(\forall j<n.\ i\neq j \longrightarrow {}^{a}Y!j \leq {}^{o}Y!j) \wedge {}^{o}X!i = {}^{a}X!i \wedge$
$^{o}Y!i = {}^{a}Y!i \wedge length\ {}^{o}X = length\ {}^{a}X \wedge length\ {}^{o}Y = length\ {}^{a}Y$⦄,
⦃$(\forall j<n.\ i\neq j \longrightarrow {}^{o}X!j = {}^{a}X!j \wedge {}^{o}Y!j = {}^{a}Y!j) \wedge$
$^{a}Y!i \leq {}^{o}Y!i \wedge length\ {}^{o}X = length\ {}^{a}X \wedge length\ {}^{o}Y = length\ {}^{a}Y$⦄,
⦃$(\acute{}X!i)\ mod\ n{=}i \wedge (\forall j<\acute{}X!i.\ j\ mod\ n{=}i \longrightarrow \neg P(B!j)) \wedge (\acute{}Y!i<m \longrightarrow P(B!(\acute{}Y!i)) \wedge \acute{}Y!i\leq m{+}i) \wedge (\exists j<n.\ \acute{}Y!j \leq \acute{}X!i)$ ⦄) *COEND*)
SAT [⦃$n<length\ \acute{}X \wedge n<length\ \acute{}Y \wedge (\forall i<n.\ \acute{}X!i{=}i \wedge \acute{}Y!i{=}m{+}i)$ ⦄,
⦃$^{o}X{=}{}^{a}X \wedge {}^{o}Y{=}{}^{a}Y$⦄,
⦃*True*⦄,
⦃$\forall i<n.\ (\acute{}X!i)\ mod\ n{=}i \wedge (\forall j<\acute{}X!i.\ j\ mod\ n{=}i \longrightarrow \neg P(B!j)) \wedge$
$(\acute{}Y!i<m \longrightarrow P(B!(\acute{}Y!i)) \wedge \acute{}Y!i\leq m{+}i) \wedge (\exists j<n.\ \acute{}Y!j \leq \acute{}X!i)$⦄]
⟨*proof*⟩

end
theory *Hoare-Parallel*
imports *OG-Examples Gar-Coll Mul-Gar-Coll RG-Examples*
begin

end

Bibliography

[1] Leonor Prensa Nieto. *Verification of Parallel Programs with the Owicki-Gries and Rely-Guarantee Methods in Isabelle/HOL*. PhD thesis, Technische Universität München, 2002.

[2] Tobias Nipkow and Leonor Prensa Nieto. Owicki/Gries in Isabelle/HOL. In J.-P. Finance, editor, *Fundamental Approaches to Software Engineering (FASE'99)*, volume 1577 of *LNCS*, pages 188–203. Springer, 1999.

[3] Leonor Prensa Nieto. The Rely-Guarantee method in Isabelle/HOL. In P. Degano, editor, *European Symposium on Programming (ESOP'03)*, volume 2618, pages 348–362, 2003.

[4] Leonor Prensa Nieto and Javier Esparza. Verifying single and multi-mutator garbage collectors with Owicki/Gries in Isabelle/HOL. In M. Nielsen and B. Rovan, editors, *Mathematical Foundations of Computer Science (MFCS 2000)*, volume 1893 of *LNCS*, pages 619–628. Springer-Verlag, 2000.