MACHINE m4
REFINES m3
SEES c4_srequest
VARIABLES

capt M1
objt M1
pct1 M1
rdpt M2
lcke M2
lck M2
pct2 M2
rsp M3
rel M3
wrt M3
wrte M3
prd M3
pct3 M3
req M4
rep M4
rqs M4
rqts M4
rqst M4
ppt M4
rpt M4
lan M4
pct4 M4

INVARIANTS

inv1: $req \subseteq REQ$
Global request messages.
inv2: $rep \subseteq REP$
Global reply messages.
inv3: $rqs \subseteq RQS$
Global special request messages.
inv4: $rqts \in AGT \to \mathbb{P}(RES)$
Local special request message copies.
inv5: $rqst \in AGT \to \mathbb{P}(RES)$
Local request message copies.
inv6: $ppt \in RES \to \mathbb{N}$
Promised pointer.
inv7: $rpt \in RES \nrightarrow \mathbb{N}$
Read pointer.
inv8: $lan \in RES \to (\mathbb{N} \nrightarrow AGT)$
Distributed lane.
inv9: $pct4 \in AGT \to AST$
Agent program counter.

EVENTS

Initialisation ⟨extended⟩

begin

act1: $capt := AGT \times \{\varnothing\}$
act2: $objt :| objt' \in AGT \to OBJ$
Every agent has an objective to begin with.
act3: $pct1 := AGT \times \{CONSUME\}$
Every agent starts with program counter at consume (@ m1).
act4: $rdpt := RES \times \{\varnothing\}$

act5: $lck := \varnothing$
act6: $lcke := AGT \times \{\varnothing\}$
act7: $pct2 := AGT \times \{LOCK\}$
act8: $rsp := \varnothing$
act9: $rel := \varnothing$
act10: $wrt := \varnothing$
act11: $wrte := AGT \times \{\varnothing\}$
act12: $prd := \varnothing$
act13: $pct3 := AGT \times \{WRITE\}$
act14: $req := \varnothing$
act15: $rep := \varnothing$
act16: $rqs := \varnothing$
act17: $rqts := AGT \times \{\varnothing\}$
act18: $rqst := AGT \times \{\varnothing\}$
act19: $ppt := resin$
act20: $rpt := \varnothing$
act21: $lan := RES \times \{\varnothing\}$
act22: $pct4 := AGT \times \{REQUEST\}$

end

Event agent_release_c ⟨ordinary⟩ $\widehat{=}$
EVT:RELEASE

extends agent_release_c

any

ag
ob

where

grd1: $capt(ag) = \varnothing$
reset capture variable
grd2: $ob \in OBJ$
take (new) objective
grd3: $pct1(ag) = RELEASE$
program counter must be at release
grd4: $pct2(ag) = RELEASE$
program counter must be at release
grd5: $pct3(ag) = RELEASE$
grd6: $pct4(ag) = RELEASE$

then

act1: $pct1(ag) := CONSUME$
change program counter to consume
act2: $objt(ag) := ob$
update agent objective
act3: $pct2(ag) := LOCK$
change program counter to lock
act4: $pct3(ag) := WRITE$
update program counter to write
act5: $pct4(ag) := REQUEST$
update program counter to wrte

end

Event agent_release_p ⟨ordinary⟩ $\widehat{=}$
EVT:RELEASE

extends agent_release_p

any

rs
ag
rl

where

grd1: $ag \in dom(capt)$
an active agent

grd2: $rs \in capt(ag)$
already captured resource
grd3: $pct1(ag) = RELEASE$
program counter at release
grd4: $pct2(ag) = RELEASE$
program counter at release
grd5: $rl \in REL \setminus rel$
create new release message
grd6: $rels(rl) = ag$
release message source is agent - ag
grd7: $reld(rl) = rs$
release message destination is resource - rs
grd8: $pct3(ag) = RELEASE$
program counter must be at release
grd9: $pct4(ag) = RELEASE$

then

act1: $capt(ag) := capt(ag) \setminus \{rs\}$
remove captured resource
act2: $lcke(ag) := lcke(ag) \setminus \{rs\}$
remove local lock message copies
act3: $rel := rel \cup \{rl\}$
create release message
act4: $lan(reld(rl)) := lan(reld(rl)) \rhd \{rels(rl)\}$

end

Event agent_consume_c ⟨ordinary⟩ $\widehat{=}$
EVT:CONSUME

extends agent_consume_c

any

ag

where

grd1: $capt(ag) = objr(objt(ag))$
objective of an agent has been fulfilled
grd2: $pct1(ag) = CONSUME$
program counter must be at consume
grd4: $pct2(ag) = CONSUME$
program counter must be at consume
grd5: $pct3(ag) = CONSUME$
program counter must be at consume
grd6: $pct4(ag) = CONSUME$

then

act1: $pct1(ag) := RELEASE$
update program counter
act2: $pct2(ag) := RELEASE$
update proram counter
act3: $pct3(ag) := RELEASE$
update program counter
act4: $pct4(ag) := RELEASE$

end

Event agent_consume_p ⟨ordinary⟩ $\widehat{=}$
EVT:CONSUME

extends agent_consume_p

any

rs
ag
cf

where

grd1: $ag \in dom(capt)$
an active agent

grd2: $rs \in objr(objt(ag))$
resource to consume has to be within objective
grd3: $rs \notin union(ran(capt))$
resource to consume cannot be captured already (by any agent)
grd4: $pct1(ag) = CONSUME$
program counter must be at consume
grd5: $pct2(ag) = CONSUME$
grd6: $rsps(cf) = rs$
to remove correct response message
grd7: $rspd(cf) = ag$
to remove correct response message
grd8: $rspn(cf) = READY$
to remove correct response message
grd9: $pct3(ag) = CONSUME$
grd10: $pct4(ag) = CONSUME$

then

act1: $capt(ag) := capt(ag) \cup \{rs\}$
act2: $rsp := rsp \setminus \{cf\}$
remove response message

end

Event resource_preready_release1 ⟨ordinary⟩ $\widehat{=}$
EVT:UNLOCK
CREATE PRE-READY MESSAGE

extends resource_pready_release1

any

rl

where

grd1: $rl \in rel$
take a sent release message
grd2: $lan(reld(rl)) = \varnothing$
updating read pointer, if there are other agents

then

act1: $rdpt(reld(rl)) := rdpt(reld(rl)) \setminus \{rels(rl)\}$
unlock read pointer
act2: $rel := rel \setminus \{rl\}$
remove release message
act3: $rpt(reld(rl)) := rpt(reld(rl)) + 1$
update read pointer

end

Event resource_preready_release2 ⟨ordinary⟩ $\widehat{=}$
EVT:UNLOCK
CREATE PRE-READY MESSAGE

extends resource_pready_release2

any

rl
pr
wr

where

grd1: $rl \in rel$
take a sent release message
grd2: $pr \in PRD \setminus prd$
grd3: $prds(pr) = reld(rl)$
grd4: $wr \in wrt$
grd5: $prdd(pr) = lan(reld(rl))(min(dom(lan(reld(rl)))))$
grd6: $lan(reld(rl)) \neq \varnothing$
updating read pointer, if there are not other agents
grd7: $wr \in wrt \wedge prdd(pr) = wrts(wr) \wedge prds(pr) = wrtd(wr) \wedge wrtn(wr) = min(dom(lan(reld(rl))))$

then

act1: $rdpt(reld(rl)) := rdpt(reld(rl)) \setminus \{rels(rl)\}$
unlock read pointer

act2: $rel := rel \setminus \{rl\}$
remove release message
act3: $prd := prd \cup \{pr\}$
act4: $wrt := wrt \setminus \{wr\}$
act5: $rpt(reld(rl)) := min(dom(lan(reld(rl))))$
update read pointer

end

Event resource_preready_release3 ⟨ordinary⟩ $\widehat{=}$
EVT:UNLOCK
CREATE PRE-READY MESSAGE

extends resource_pready_release3

any
rl
pr
wr

where
grd1: $rl \in rel$
take a sent release message
grd2: $pr \in PRD \setminus prd$
grd3: $prds(pr) = reld(rl)$
grd4: $prdd(pr) = lan(reld(rl))(min(dom(lan(reld(rl)))))$
grd5: $lan(reld(rl)) \neq \varnothing$
updating read pointer, if there are not other agents
grd6: $wr \notin wrt \wedge prdd(pr) = wrts(wr) \wedge prds(pr) = wrtd(wr) \wedge wrtn(wr) = min(dom(lan(reld(rl))))$

then
act1: $rdpt(reld(rl)) := rdpt(reld(rl)) \setminus \{rels(rl)\}$
unlock read pointer
act2: $rel := rel \setminus \{rl\}$
remove release message
act3: $prd := prd \cup \{pr\}$
act4: $rpt(reld(rl)) := min(dom(lan(reld(rl))))$
update read pointer

end

Event agent_unlock_c ⟨ordinary⟩ $\widehat{=}$
EVT:UNLOCK

extends agent_unlock_c

any
ag

where
grd1: $lcke(ag) = \varnothing$
unlock is completed when no local copies of lock have been left
grd2: $pct2(ag) = UNLOCK$
and program counter must be at unlock
grd3: $pct3(ag) = UNLOCK$
grd4: $pct4(ag) = UNLOCK$

then
act1: $pct2(ag) := LOCK$
update program counter to lock
act2: $pct3(ag) := CONFIRMP$
act3: $pct4(ag) := CONFIRMP$

end

Event agent_unlock_p ⟨ordinary⟩ $\widehat{=}$
EVT:UNLOCK

extends agent_unlock_p

any
cf
rl unlock

ms

where

grd1: $cf \in rsp$
take a received confirm message

grd2: $rl \in REL \setminus rel$
created new unlock message

grd3: $reld(rl) = rsps(cf)$
release message destination, must be the soure of respond message

grd4: $rels(rl) = rspd(cf)$
release message source, must be the destination of repond message

grd5: $rspn(cf) = READY \Rightarrow ms = \{rl\}$
confirm message receive must be CONFIRM

grd6: $rspn(cf) = DENY \Rightarrow ms = \varnothing$
if message is deny do nothing, just remove that message

grd7: $pct2(rspd(cf)) = UNLOCK$
the program counter of the cnfd(cf) must be at UNLOCK

grd8: $pct3(rspd(cf)) = UNLOCK$
program counte must be at at unlock

grd9: $pct4(rspd(cf)) = UNLOCK$
program counter must be at unlock

then

act1: $rel := rel \cup ms$
send new release message

act2: $rsp := rsp \setminus \{cf\}$
delete confirm message

act3: $lcke(rspd(cf)) := lcke(rspd(cf)) \setminus \{rsps(cf)\}$
remove locally saved lock message

end

Event agent_decide ⟨ordinary⟩ $\widehat{=}$

extends agent_decide

any

ag

pc

where

grd1: $rsps[rsp \cap rspd^{-1}[\{ag\}]] = lcke(ag)$
confirm that all lock messages have been replied with confirm (ready/deny) messages

grd2: $rsps[rsp \cap rspd^{-1}[\{ag\}]] = objr(objt(ag))$
confirm that objective has been fulfilled

grd3: $rspn[rsp \cap rspd^{-1}[\{ag\}]] = \{READY\} \Rightarrow pc = CONSUME$
if all messages are ready then update progam counter to consume

grd4: $DENY \in rspn[rsp \cap rspd^{-1}[\{ag\}]] \Rightarrow pc = UNLOCK$
if exists deny message then update program counter to unlock

grd5: $pct2(ag) = CONFIRMC$
program counter mus be at confirmc

grd6: $pct3(ag) = CONFIRMC$

grd7: $pct4(ag) = CONFIRMC$

then

act1: $pct2(ag) := pc$
update with new program counter value

act2: $pct3(ag) := pc$
update program counter

act3: $pct4(ag) := pc$
update program counter value

end

Event resource_respond ⟨ordinary⟩ $\widehat{=}$
EVT:RESPOND

extends resource_respond

any

lc

re
rp

where

grd1: $lc \in lck$
take a lock message
grd2: $re \in RSP \setminus rsp$
create new confirm message
grd3: $rsps(re) = lckd(lc)$
grd4: $rspd(re) = lcks(lc)$
grd5: $rdpt(rsps(re)) = \varnothing \Rightarrow rspn(re) = READY \land rp = \{rspd(re)\}$
if resource's read pointer is not locked, lock it and send ready message to the agent
grd6: $rdpt(rsps(re)) \neq \varnothing \Rightarrow rspn(re) = DENY \land rp = rdpt(rsps(re))$
if resource's read pointer is locked, send deny message and remain locked

then

act1: $rsp := rsp \cup \{re\}$
create confirm message
act2: $lck := lck \setminus \{lc\}$
remove received lock message
act3: $rdpt(rsps(re)) := rp$
update read pointer (or else stick the same)

end

Event agent_lock_c ⟨ordinary⟩ $\widehat{=}$
EVT:LOCK

extends agent_lock_c

any

ag

where

grd1: $lcke(ag) = objr(objt(ag))$
objective has been fulfilled with lock messages
grd2: $pct2(ag) = LOCK$
program counter at lock
grd3: $pct3(ag) = LOCK$
program counter at lock
grd4: $pct4(ag) = LOCK$

then

act1: $pct2(ag) := CONFIRMC$
next step is to confirm and decide whether consume/unlock
act2: $pct3(ag) := CONFIRMC$
act3: $pct4(ag) := CONFIRMC$

end

Event agent_lock_p ⟨ordinary⟩ $\widehat{=}$
EVT:LOCK

extends agent_lock_p

any

lc
pr

where

grd1: $lc \in LCK \setminus lck$
create a new lock message
grd2: $lckd(lc) \notin lcke(lcks(lc))$
message must not be sent already
grd3: $lckd(lc) \in objr(objt(lcks(lc)))$
lock message must be within the objective
grd4: $pct2(lcks(lc)) = LOCK$
program counter must be at lock
grd5: $pct3(lcks(lc)) = LOCK$
program counter at lock
grd6: $pr \in prd$
take a received pre-ready message

grd7: $lcks(lc) = prdd(pr)$
grd8: $lckd(lc) = prds(pr)$
grd9: $pct4(lcks(lc)) = LOCK$

then

act1: $lck := lck \cup \{lc\}$
send new lock message
act2: $lcke(lcks(lc)) := lcke(lcks(lc)) \cup \{lckd(lc)\}$
save a copy locally
act3: $prd := prd \setminus \{pr\}$
delete a pre-ready message
act4: $wrte(lcks(lc)) := wrte(lcks(lc)) \setminus \{lckd(lc)\}$
remove locally saved lock messages

end

Event agent_pready_confirm ⟨ordinary⟩ $\widehat{=}$

extends agent_pready_confirm

any

ag

where

grd1: $prds[prd \cap prdd^{-1}[\{ag\}]] = objr(objt(ag))$
objective must be fulfilled with pre-ready messages
grd2: $pct3(ag) = CONFIRMP$
program counter must be at comfirm pre-ready
grd4: $pct4(ag) = CONFIRMP$

then

act1: $pct3(ag) := LOCK$
update program counter to lock
act2: $pct4(ag) := LOCK$

end

Event resource_pready_write ⟨ordinary⟩ $\widehat{=}$
EVT:PRE-READY

extends resource_pready_write

any

wr
pr

where

grd1: $wr \in wrt$
take a write message
grd2: $pr \in PRD \setminus prd$
create new pre-ready message
grd3: $rdpt(wrtd(wr)) = \varnothing$
send pre-ready only if read pointer is not locked (taken by other agent)
grd4: $prdd(pr) = wrts(wr)$
grd5: $prds(pr) = wrtd(wr)$
grd6: $rpt(prds(pr)) = wrtn(wr)$
if read pointer points at your lane

then

act1: $prd := prd \cup \{pr\}$
create new pre-ready message
act2: $wrt := wrt \setminus \{wr\}$
delete write message

end

Event agent_write_c ⟨ordinary⟩ $\widehat{=}$
EVT:WRITE

extends write_complete

any

ag

where

grd1: $ag \in dom(wrte)$
an active agent
grd2: $wrte(ag) = objr(objt(ag))$
an objective has been fulfiled
grd3: $pct3(ag) = WRITE$
program counter at write
grd4: $pct4(ag) = WRITE$

then

act1: $pct3(ag) := CONFIRMP$
writting completed, proceed to consuming
act2: $pct4(ag) := CONFIRMP$

end

Event agent_write_p ⟨ordinary⟩ $\widehat{=}$
EVT:WRITE

extends agent_write_p

any

wr
rp
ag

where

grd1: $wr \in WRT \setminus wrt$
generate a new write message
grd2: $wrtd(wr) \notin wrte(wrts(wr))$
cannot send the same content twice
grd3: $wrtd(wr) \in objr(objt(wrts(wr)))$
write just to resources which belong to your objective
grd4: $pct3(wrts(wr)) = WRITE$
program counter must be at write
grd5: $pct4(wrts(wr)) = WRITE$
grd6: $ag \in repd[rep]$
grd7: $rp \in rep$
grd8: $rp \in repd^{-1}[\{ag\}]$
grd9: $wrtd(wr) = reps(rp)$
grd10: $wrts(wr) = repd(rp)$
grd11: $wrtn(wr) = repn(rp)$

then

act1: $wrt := wrt \cup \{wr\}$
add a write message
act2: $wrte(wrts(wr)) := wrte(wrts(wr)) \cup \{wrtd(wr)\}$
store a local copy of write message
act3: $rep := rep \setminus \{rp\}$
remove reply message
act4: $rqts(wrts(wr)) := rqts(wrts(wr)) \setminus \{wrtd(wr)\}$
remove local request
act5: $rqst(wrts(wr)) := rqst(wrts(wr)) \setminus \{wrtd(wr)\}$
remove local special requests
act6: $lan(wrtd(wr)) := lan(wrtd(wr)) \lhd \{wrtn(wr) \mapsto ag\}$
create a lane
act7: $rpt(wrtd(wr)) := min(dom(lan(wrtd(wr)) \lhd \{wrtn(wr) \mapsto ag\}))$
update read pointer

end

Event agent_renegotiate_c ⟨ordinary⟩ $\widehat{=}$
EVT:RENEGOTIATE

any

ag

where

grd1: $card(repn[rep \cap repd^{-1}[\{ag\}]]) = 1$
all replies must contain the same index

grd2: $reps[rep \cap repd^{-1}[\{ag\}]] = objr(objt(ag))$
agent objective must be fulfilled by reply messages
grd3: $pct4(ag) = RENEGOTIATE$

then

act1: $pct4(ag) := WRITE$

end

Event agent_renegotiate_p ⟨ordinary⟩ $\widehat{=}$
EVT:RENEGOTIATE

any

ag agent
rp replies of the interest
rq generated special requests

where

grd1: $ag \in repd[rep]$
an active agent
grd2: $pct4(ag) = RENEGOTIATE$
grd3: $rp \in dom(((rep \cap repd^{-1}[\{ag\}]) \lhd repn) \rhd \{max(repn[rep \cap repd^{-1}[\{ag\}]])\})$
reply message subpool with slot number lower than maximum
grd4: $rq \in RQS \setminus rqs$
generate special request messages
grd5: $rqss(rq) = ag$
the source of special request messages must be the same
grd6: $rqsd(rq) = reps(rp)$
special request messages must be sent to all resources which previously replied with with slot number smaller than maximum.
grd7: $rqsn(rq) = max(repn[rep \cap repd^{-1}[\{ag\}]])$
special request slot number is the maximum of all reply messages

then

act1: $rep := rep \setminus \{rp\}$
delete replies
act2: $rqs := rqs \cup \{rq\}$
generate special requests
act4: $rqts(rqss(rq)) := rqts(rqss(rq)) \cup \{rqsd(rq)\}$
generate special requests
act5: $rqst(rqss(rq)) := rqst(rqss(rq)) \setminus \{rqsd(rq)\}$
remove request

end

Event agent_confirm_write_renegotiate ⟨ordinary⟩ $\widehat{=}$
EVT:CONFIRM-WRITE-RENEGOTIATE

any

ag agent
pc program counter

where

grd1: $ag \in repd[rep]$
an active agent
grd2: $card(rqst(ag) \cup rqts(ag)) = card(rep \cap repd^{-1}[\{ag\}])$
all request and special_request messages have been replied
grd3: $reps[rep \cap repd^{-1}[\{ag\}]] = objr(objt(ag))$
an objective has been fulfiled with reply messsages
grd4: $pct4(ag) = CONFIRMW$
program counter at confirmw
grd5: $card(repn[rep \cap repd^{-1}[\{ag\}]]) > 1 \Rightarrow pc = RENEGOTIATE$
if reply indedexes heterogeneous then renegotiate renegotiate (update program counter)
grd6: $card(repn[rep \cap repd^{-1}[\{ag\}]]) = 1 \Rightarrow pc = WRITE$
if reply indexes homogeneous then write (update program counter)

then

act1: $pct4(ag) := pc$
update program counter

end

Event resource_reply_special ⟨ordinary⟩ $\widehat{=}$
EVT:REPLY

any
rq
rp
where
grd1: $rq \in rqs$
special request message
grd2: $rp \in REP \setminus rep$
new reply message
grd3: $repd(rp) = rqss(rq)$
destination of the reply message is the source of the request message
grd4: $reps(rp) = rqsd(rq)$
source of the reply message is the destination of the request message
grd5: $repn(rp) = max(\{ppt(reps(rp)), rqsn(rq)\})$
reply slot number is the maximum of promised pointer and special request slot number
then
act1: $rep := rep \cup \{rp\}$
generate a reply
act2: $rqs := rqs \setminus \{rq\}$
delete special request message
act3: $ppt(reps(rp)) := max(\{ppt(reps(rp)), rqsn(rq)\}) + 1$
increment promised pointer
end

Event resource_reply_general ⟨ordinary⟩ $\widehat{=}$
EVT:REPLY

any
rp
rq
where
grd1: $rq \in req$
take a request message
grd2: $rp \in REP \setminus rep$
generate a new reply message
grd3: $repd(rp) = reqs(rq)$
the destination of reply message is the source of the request message
grd4: $reps(rp) = reqd(rq)$
the source of reply message is the destination of the request message
grd5: $repn(rp) = ppt(reps(rp))$
reply slot number is the promised pointer
then
act1: $rep := rep \cup \{rp\}$
generate a reply message
act2: $req := req \setminus \{rq\}$
delete the request message
act3: $ppt(reps(rp)) := ppt(reps(rp)) + 1$
increment promised pointer
end

Event agent_request_c ⟨ordinary⟩ $\widehat{=}$
EVT: REUQEST

any
ag
where
grd1: $ag \in dom(rqst)$
grd2: $rqst(ag) \cup rqts(ag) = objr(objt(ag))$
all request messages have been sent (objective fulfilled)
grd3: $pct4(ag) = REQUEST$
program counter at request

then

act1: $pct4(ag) := CONFIRMW$

update program counter

end

Event agent_request_p ⟨ordinary⟩ $\widehat{=}$
EVT:REQUEST

any

rq

where

grd1: $rq \in REQ \setminus req$

take a new request message

grd2: $reqd(rq) \notin rqst(reqs(rq))$

request message cannot be sent already (grd1 is not enough, since we allow multiple messages to have the same source/destination)

grd3: $reqd(rq) \notin rqts(reqs(rq))$

request message cannot be sent already (special request)

grd4: $reqd(rq) \in objr(objt(reqs(rq)))$

requested resource must be within objective

grd5: $pct4(reqs(rq)) = REQUEST$

program counter at request

then

act1: $req := req \cup \{rq\}$

create a request message

act2: $rqst(reqs(rq)) := rqst(reqs(rq)) \cup \{reqd(rq)\}$

local copy of the request

end

END