PiCore: A Rely-guarantee Framework for Event-based Systems

Yongwang Zhao
School of Computer Science and Engineering, Beihang University, China
zhaoyongwang@gmail.com, zhaoyw@buaa.edu.cn

March 17, 2019

Contents

1 Integrating the SIMP language into Picore

theory *SIMP-plus*
imports *HOL−Hoare-Parallel.RG-Hoare*
begin

inductive *rghoare-p* :: $[{'a}\ com\ option,\ {'a}\ set,\ ({'a} \times {'a})\ set,\ ({'a} \times {'a})\ set,\ {'a}\ set] \Rightarrow bool$
$(\vdash_I\ \text{-}\ sat_p\ [\text{-}, \text{-}, \text{-}, \text{-}]\ [60,0,0,0,0]\ 45)$
where

Basic: $[\![\ pre \subseteq \{s.\ f\ s \in post\};\ \{(s,t).\ s \in pre \wedge (t=f\ s)\} \subseteq guar;$
$stable\ pre\ rely;\ stable\ post\ rely\ ]\!]$
$\Longrightarrow \vdash_I Some\ (Basic\ f)\ sat_p\ [pre,\ rely,\ guar,\ post]$

| *Seq*: $[\![\ \vdash_I Some\ P\ sat_p\ [pre,\ rely,\ guar,\ mid];\ \vdash_I Some\ Q\ sat_p\ [mid,\ rely,\ guar,\ post]\ ]\!]$
$\Longrightarrow \vdash_I Some\ (Seq\ P\ Q)\ sat_p\ [pre,\ rely,\ guar,\ post]$

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

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

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

| *None-hoare*: $[\![\ stable\ pre\ rely;\ pre \subseteq post\ ]\!] \Longrightarrow \vdash_I None\ sat_p\ [pre,\ rely,\ guar,\ post]$

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

| *Unprecond*: $[\![\ \vdash_I P\ sat_p\ [pre,\ rely,\ guar,\ post];\ \vdash_I P\ sat_p\ [pre',\ rely,\ guar,\ post]\ ]\!]$
$\Longrightarrow \vdash_I P\ sat_p\ [pre \cup pre',\ rely,\ guar,\ post]$

| *Intpostcond*: $[\![\ \vdash_I P\ sat_p\ [pre,\ rely,\ guar,\ post];\ \vdash_I P\ sat_p\ [pre,\ rely,\ guar,\ post']\ ]\!]$
$\Longrightarrow \vdash_I P\ sat_p\ [pre,\ rely,\ guar,\ post \cap post']$

| *Allprecond*: $\forall v \in U.\ \vdash_I P\ sat_p\ [\{v\},\ rely,\ guar,\ post]$
$\Longrightarrow \vdash_I P\ sat_p\ [U,\ rely,\ guar,\ post]$

| *Emptyprecond*: $\vdash_I$ *P sat*$_p$ [{}, *rely*, *guar*, *post*]

definition *prog-validity* :: *'a com option* ⇒ *'a set* ⇒ (*'a* × *'a*) *set* ⇒ (*'a* × *'a*) *set* ⇒ *'a set* ⇒ *bool*
($\models_I$ - *sat*$_p$ [-, -, -, -] [*60*,*0*,*0*,*0*,*0*] *45*) **where**
$\models_I$ *P sat*$_p$ [*pre*, *rely*, *guar*, *post*] ≡
∀ *s*. *cp P s* ∩ *assum*(*pre*, *rely*) ⊆ *comm*(*guar*, *post*)

1.1 lemmas of SIMP

lemma *etran-or-ctran2-disjI3*:
⟦ *x*∈*cptn*; *Suc i*<*length x*; ¬ *x*!*i* −*c*→ *x*!*Suc i*⟧ ⟹ *x*!*i* −*e*→ *x*!*Suc i*
apply(*induct x arbitrary*:*i*)
apply *simp*
apply *clarify*
apply(*rule cptn.cases*)
apply *simp*+
using *less-Suc-eq-0-disj etran.intros* **apply** *force*
apply(*case-tac i*,*simp*)
by *simp*

lemma *stable-id*: *stable P Id*
unfolding *stable-def Id-def* **by** *auto*

lemma *stable-id2*: *stable P* {(*s*,*t*). *s* = *t*}
unfolding *stable-def* **by** *auto*

lemma *stable-int2*: *stable s r* ⟹ *stable t r* ⟹ *stable* (*s* ∩ *t*) *r*
by (*metis* (*full-types*) *IntD1 IntD2 IntI stable-def*)

lemma *stable-int3*: *stable k r* ⟹ *stable s r* ⟹ *stable t r* ⟹ *stable* (*k* ∩ *s* ∩ *t*) *r*
by (*metis* (*full-types*) *IntD1 IntD2 IntI stable-def*)

lemma *stable-un2*: *stable s r* ⟹ *stable t r* ⟹ *stable* (*s* ∪ *t*) *r*
by (*simp add*: *stable-def*)

lemma *Seq2*: ⟦ $\vdash_I$ *Some P sat*$_p$ [*pre*, *rely*, *guar*, *mida*]; *mida* ⊆ *midb*; $\vdash_I$ *Some Q sat*$_p$ [*midb*, *rely*, *guar*, *post*] ⟧
⟹ $\vdash_I$ *Some* (*Seq P Q*) *sat*$_p$ [*pre*, *rely*, *guar*, *post*]
using *Seq*[*of P pre rely guar mida Q post*]
Conseq[*of mida midb rely rely guar guar post post*]
by *blast*

1.2 Soundness of the Rule of Consequence

lemma *Conseq-sound*:
  $[\![pre \subseteq pre'; rely \subseteq rely'; guar' \subseteq guar; post' \subseteq post;$
  $\models_I P\ sat_p\ [pre', rely', guar', post']]\!]$
  $\Longrightarrow \models_I P\ sat_p\ [pre, rely, guar, post]$
apply(*simp add:prog-validity-def assum-def comm-def*)
apply *clarify*
apply(*erule-tac x=s* **in** *allE*)
apply(*drule-tac c=x* **in** *subsetD*)
 apply *force*
apply *force*
done

1.3 Soundness of the Rule of Unprecond

lemma *Unprecond-sound*:
  assumes *p0*: $\models_I P\ sat_p\ [pre, rely, guar, post]$
    and *p1*: $\models_I P\ sat_p\ [pre', rely, guar, post]$
   shows $\models_I P\ sat_p\ [pre \cup pre', rely, guar, post]$
proof −
{
  fix *s c*
  assume $c \in cp\ P\ s \cap assum(pre \cup pre', rely)$
  hence *a1*: $c \in cp\ P\ s$ **and**
      a2: $c \in assum(pre \cup pre', rely)$ **by** *auto*
  hence $c \in assum(pre, rely) \vee c \in assum(pre', rely)$
   by (*metis* (*no-types*, *lifting*) *CollectD CollectI Un-iff assum-def prod.simps*(*2*))
  hence $c \in comm(guar, post)$
    proof
      assume $c \in assum\ (pre, rely)$
      with *p0 a1* **show** $c \in comm\ (guar, post)$
        unfolding *prog-validity-def* **by** *auto*
    next
      assume $c \in assum\ (pre', rely)$
      with *p1 a1* **show** $c \in comm\ (guar, post)$
        unfolding *prog-validity-def* **by** *auto*
    qed
}
then show *?thesis* **unfolding** *prog-validity-def* **by** *auto*
qed

1.4 Soundness of the Rule of Intpostcond

lemma *Intpostcond-sound*:
  assumes *p0*: $\models_I P\ sat_p\ [pre, rely, guar, post]$
    and *p1*: $\models_I P\ sat_p\ [pre, rely, guar, post']$
   shows $\models_I P\ sat_p\ [pre, rely, guar, post \cap post']$
proof −
{

fix *s c*
assume *a0*: $c \in cp\ P\ s \cap assum(pre,\ rely)$
with *p0* have $c \in comm(guar,\ post)$ unfolding *prog-validity-def* by *auto*
moreover
from *a0 p1* have $c \in comm(guar,\ post')$ unfolding *prog-validity-def* by *auto*
ultimately have $c \in comm(guar,\ post \cap post')$
by (*simp add*: *comm-def*)
}
then show *?thesis* unfolding *prog-validity-def* by *auto*
qed

1.5 Soundness of the Rule of Allprecond

lemma *Allprecond-sound*:
assumes *p1*: $\forall v{\in}U.\ \models_I P\ sat_p\ [\{v\},\ rely,\ guar,\ post]$
shows $\models_I P\ sat_p\ [U,\ rely,\ guar,\ post]$
proof −
{
fix *s c*
assume *a0*: $c \in cp\ P\ s \cap assum(U,\ rely)$
then obtain *x* **where** *a1*: $x{\in}U \wedge snd\ (c!0) = x$
by (*metis* (*no-types*, *lifting*) *CollectD IntD2 assum-def prod.simps*(*2*))

with *p1* **have** $\models_I P\ sat_p\ [\{x\},\ rely,\ guar,\ post]$ **by** *simp*
hence *a2*: $\forall s.\ cp\ P\ s \cap assum(\{x\},\ rely) \subseteq comm(guar,\ post)$ **unfolding** *prog-validity-def* **by** *simp*

from *a0* **have** $c \in assum(U,\ rely)$ **by** *simp*
hence $snd\ (c!0) \in U \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-e}{\rightarrow}\ c!(Suc\ i) \longrightarrow (snd\ (c!i),\ snd\ (c!Suc\ i)) \in rely)$ **by** (*simp add*:*assum-def*)
with *a1* **have** $snd\ (c!0) \in \{x\} \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-e}{\rightarrow}\ c!(Suc\ i) \longrightarrow (snd\ (c!i),\ snd\ (c!Suc\ i)) \in rely)$ **by** *simp*

hence $c{\in}assum(\{x\},\ rely)$ **by** (*simp add*:*assum-def*)
with *a0 a2* **have** $c \in comm(guar,\ post)$ **by** *auto*
}
then show *?thesis* **using** *prog-validity-def* **by** *blast*
qed

1.6 Soundness of the Rule of Emptyprecond

lemma *Emptyprecond-sound*: $\models_I P\ sat_p\ [\{\},\ rely,\ guar,\ post]$
unfolding *prog-validity-def* **by**(*simp add*:*assum-def*)

1.7 Soundness of None rule

lemma *none-all-none*: $c!0{=}(None,s) \wedge c \in cptn \Longrightarrow \forall i{<}length\ c.\ fst\ (c\ !\ i) = None$
proof(*induct c arbitrary*:*s*)

case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a c*)
assume *p1*: $\bigwedge s.\ c\ !\ 0 = (None,\ s) \wedge c \in cptn \Longrightarrow \forall i{<}length\ c.\ fst\ (c\ !\ i) = None$
and *p2*: $(a\ \#\ c)\ !\ 0 = (None,\ s) \wedge a\ \#\ c \in cptn$
hence *a0*: $a = (None,s)$ **by** *simp*
thus *?case*
proof(*cases* $c = []$)
case *True*
with *a0* **show** *?thesis* **by** *auto*
next
case *False*
assume *b0*: $c \neq []$
with *p2* **have** *c-cpts*: $c \in cptn$ **using** *tl-in-cptn* **by** *fast*
from *b0* **obtain** c' **and** b **where** *bc'*: $c = b\ \#\ c'$
using *list.exhaust* **by** *blast*
from *a0* **have** $\neg\ a\ {-c\rightarrow}\ b$ **by**(*force elim*: *ctran.cases*)
with *p2* **have** $a\ {-e\rightarrow}\ b$ **using** *bc' etran-or-ctran2-disjI3*[*of a#c 0*] **by** *auto*
hence $fst\ b = None$ **using** *etran.cases*
by (*metis a0 prod.collapse*)
with *p1 bc' c-cpts* **have** $\forall i{<}length\ c.\ fst\ (c\ !\ i) = None$
by (*metis nth-Cons-0 prod.collapse*)
with *a0* **show** *?thesis*
by (*simp add*: *nth-Cons'*)
qed

qed

lemma *None-sound-h*: $\forall x.\ x \in pre \longrightarrow (\forall y.\ (x,\ y) \in rely \longrightarrow y \in pre) \Longrightarrow$
$pre \subseteq post \Longrightarrow$
$snd\ (c\ !\ 0) \in pre \Longrightarrow$
$c \neq [] \Longrightarrow \forall i.\ Suc\ i < length\ c \longrightarrow (snd\ (c\ !\ i),\ snd\ (c\ !\ Suc\ i)) \in rely$
$\Longrightarrow i < length\ c \Longrightarrow snd\ (c\ !\ i) \in pre$
apply(*induct i*) **by** *auto*

lemma *None-sound*:
$[\![$ *stable pre rely*; $pre \subseteq post$ $]\!]$
$\Longrightarrow \models_I None\ sat_p\ [pre,\ rely,\ guar,\ post]$
proof −
assume *p0*: *stable pre rely*
and *p2*: $pre \subseteq post$
{
fix *s c*
assume *a0*: $c \in cp\ None\ s \cap assum(pre,\ rely)$
hence *c1*: $c!0{=}(None,s) \wedge c \in cptn$ **by** (*simp add*:*cp-def*)
from *a0* **have** *c2*: $snd\ (c!0) \in pre \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-e\rightarrow}\ c!(Suc\ i) \longrightarrow (snd\ (c!i),\ snd\ (c!Suc\ i)) \in rely)$

by (*simp add*:*assum-def*)
from *c1* **have** *c-ne-empty*: $c \neq []$
by *auto*
from *c1* **have** *c-all-none*: $\forall\, i<length\ c.\ fst\ (c\ !\ i) = None$ **using** *none-all-none*
by *fast*

{
fix *i*
assume *suci*: $Suc\ i<length\ c$
and *cc*: $c!i\ -c\rightarrow\ c!(Suc\ i)$
from *suci c-all-none* **have** $c!i\ -e\rightarrow\ c!(Suc\ i)$
by (*metis Suc-lessD etran.intros prod.collapse*)
with *cc* **have**$(snd\ (c!i),\ snd\ (c!Suc\ i)) \in guar$
using *c1 etran-or-ctran2-disjI1 suci* **by** *auto*
}
moreover
{
assume *last-none*: $fst\ (last\ c) = None$
from *c2 c-all-none* **have** $\forall\, i.\ Suc\ i<length\ c \longrightarrow (snd\ (c!i),\ snd\ (c!Suc\ i)) \in rely$
by (*metis Suc-lessD etran.intros prod.collapse*)
with *p0 p2 c2 c-ne-empty* **have** $\forall\, i.\ i < length\ c \longrightarrow snd\ (c\ !\ i) \in pre$
apply(*simp add*: *stable-def*) **apply** *clarify* **using** *None-sound-h* **by** *blast*
with *p2 c-ne-empty* **have** $snd\ (last\ c) \in post$
using *One-nat-def c-ne-empty last-conv-nth* **by** *force*
}
ultimately have $c \in comm(guar,\ post)$ **by** (*simp add*:*comm-def*)
}
thus $\models_I None\ sat_p\ [pre,\ rely,\ guar,\ post]$ **using** *prog-validity-def* **by** *blast*
qed

1.8 Soundness of the Await rule

lemma *Await-sound*:
$[\![stable\ pre\ rely;\ stable\ post\ rely;$
$\forall\, V.\ \vdash_I Some\ P\ sat_p\ [pre \cap b \cap \{s.\ s = V\},\ \{(s,\ t).\ s = t\},$
$UNIV,\ \{s.\ (V,\ s) \in guar\} \cap post] \wedge$
$\models_I Some\ P\ sat_p\ [pre \cap b \cap \{s.\ s = V\},\ \{(s,\ t).\ s = t\},$
$UNIV,\ \{s.\ (V,\ s) \in guar\} \cap post]\ ]\!]$
$\Longrightarrow \models_I Some\ (Await\ b\ P)\ sat_p\ [pre,\ rely,\ guar,\ post]$
apply(*unfold prog-validity-def*)
apply *clarify*
apply(*simp add*:*comm-def*)
apply(*rule conjI*)
apply *clarify*
apply(*simp add*:*cp-def assum-def*)
apply *clarify*
apply(*frule-tac* $j=0$ **and** $k=i$ **and** $p=pre$ **in** *stability*,*simp-all*)

```
   apply(erule-tac x=ia in allE,simp)
  apply(subgoal-tac x∈ cp (Some(Await b P)) s)
  apply(erule-tac i=i in unique-ctran-Await,force,simp-all)
  apply(simp add:cp-def)

 apply(erule ctran.cases,simp-all)
 apply(drule Star-imp-cptn)
 apply clarify
 apply(erule-tac x=sa in allE)
 apply clarify
 apply(erule-tac x=sa in allE)
 apply(drule-tac c=l in subsetD)
  apply (simp add:cp-def)
  apply clarify
  apply(erule-tac x=ia and P=λi. H i ⟶ (J i, I i)∈ctran for H J I in allE,simp)
  apply(erule etranE,simp)
 apply simp
apply clarify
apply(simp add:cp-def)
apply clarify
apply(frule-tac i=length x - 1 in exists-ctran-Await-None,force)
  apply (case-tac x,simp+)
 apply(rule last-fst-esp,simp add:last-length)
 apply(case-tac x, simp+)
apply clarify
apply(simp add:assum-def)
apply clarify
apply(frule-tac j=0 and k=j and p=pre in stability,simp-all)
  apply(erule-tac x=i in allE,simp)
 apply(erule-tac i=j in unique-ctran-Await,force,simp-all)
apply(case-tac x!j)
apply clarify
apply simp
apply(drule-tac s=Some (Await b P) in sym,simp)
apply(case-tac x!Suc j,simp)
apply(rule ctran.cases,simp)
apply(simp-all)
apply(drule Star-imp-cptn)
apply clarify
apply(erule-tac x=sa in allE)
apply clarify
apply(erule-tac x=sa in allE)
apply(drule-tac c=l in subsetD)
 apply (simp add:cp-def)
 apply clarify
 apply(erule-tac x=i and P=λi. H i ⟶ (J i, I i)∈ctran for H J I in allE,simp)
 apply(erule etranE,simp)
apply simp
apply clarify
```

apply(*frule-tac j=Suc j* **and** *k=length x − 1* **and** *p=post* **in** *stability*,*simp-all*)
apply(*case-tac x*,*simp+*)
apply(*erule-tac x=i* **in** *allE*)
apply(*erule-tac i=j* **in** *unique-ctran-Await*,*force*,*simp-all*)
apply *arith+*
apply(*case-tac x*)
apply(*simp add*:*last-length*)+
done

theorem *rgsound-p*:
$\vdash_I P\ sat_p\ [pre,\ rely,\ guar,\ post] \Longrightarrow \models_I P\ sat_p\ [pre,\ rely,\ guar,\ post]$
apply(*erule rghoare-p.induct*)
using *RG-Hoare.Basic-sound* **apply**(*simp add*:*prog-validity-def com-validity-def*)
apply *blast*
using *RG-Hoare.Seq-sound* **apply**(*simp add*:*prog-validity-def com-validity-def*) **apply** *blast*
using *RG-Hoare.Cond-sound* **apply**(*simp add*:*prog-validity-def com-validity-def*)
apply *blast*
using *RG-Hoare.While-sound* **apply**(*simp add*:*prog-validity-def com-validity-def*)
apply *blast*
using *Await-sound* **apply** *fastforce*
apply(*force elim*:*None-sound*)
apply(*erule Conseq-sound*,*simp+*)
apply(*erule Unprecond-sound*,*simp+*)
apply(*erule Intpostcond-sound*,*simp+*)
using *Allprecond-sound* **apply** *force*
using *Emptyprecond-sound* **apply** *force*
done

end

2 Integrating the SIMP language into Picore

theory *picore-SIMP*
imports *PiCore.PiCore-RG-Invariant SIMP-plus*
begin

print-locale *event-validity*
print-locale *event-hoare*

abbreviation *ptranI* :: $'Env \Rightarrow ('a\ conf \times 'a\ conf)\ set$
where *ptranI* $\Gamma \equiv$ *ctran*

abbreviation *petranI* :: $'Env \Rightarrow 'a\ conf \Rightarrow 'a\ conf \Rightarrow bool$
where *petranI* $\Gamma \equiv$ *etran'*

abbreviation *cpts-pI* :: $'Env \Rightarrow 'a\ confs\ set$
where *cpts-pI* $\Gamma \equiv$ *cptn*

abbreviation *cpts-of-pI* :: $'Env \Rightarrow ('a\ com)\ option \Rightarrow 'a \Rightarrow ('a\ confs)\ set$ **where**
cpts-of-pI $\Gamma \equiv cp$

abbreviation *prog-validityI* :: $'Env \Rightarrow ('a\ com)\ option \Rightarrow 'a\ set \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set \Rightarrow 'a\ set \Rightarrow bool$
where *prog-validityI* $\Gamma\ P \equiv$ *prog-validity* P

abbreviation *assume-pI* :: $'Env \Rightarrow ('a\ set \times ('a \times 'a)\ set) \Rightarrow ('a\ confs)\ set$
where *assume-pI* $\Gamma \equiv assum$

abbreviation *commit-pI* :: $'Env \Rightarrow (('a \times 'a)\ set \times 'a\ set) \Rightarrow ('a\ confs)\ set$
where *commit-pI* $\Gamma \equiv comm$

abbreviation *rghoare-pI* :: $'Env \Rightarrow [('a\ com)\ option, 'a\ set, ('a \times 'a)\ set, ('a \times 'a)\ set, 'a\ set] \Rightarrow bool$
$(\text{-} \vdash_I \text{-}\ sat_p\ [\text{-}, \text{-}, \text{-}, \text{-}]\ [60,0,0,0,0]\ 45)$
where *rghoare-pI* $\Gamma \equiv$ *rghoare-p*

lemma *cpts-pI-ne-empty*: $[] \notin$ *cpts-pI* Γ
by (*simp*)

lemma *petran-simpsI*:
petranI $\Gamma\ (a, b)\ (c, d) \Longrightarrow a = c$
by(*simp add*: *etran.simps*)

lemma *none-no-tranI'*: $((Q, s),(P,t)) \in$ *ptranI* $\Gamma \Longrightarrow Q \neq None$
apply (*simp*) **apply**(*rule ctran.cases*)
by *simp+*

lemma *none-no-tranI*: $((None, s),(P,t)) \notin$ *ptranI* Γ
using *none-no-tranI'*
by *fast*

lemma *ptran-neqI*: $((P, s),(P,t)) \notin$ *ptranI* Γ
by (*simp*)

interpretation *event ptranI petranI None*
using *petran-simpsI none-no-tranI ptran-neqI*
event.intro[*of petranI None ptranI*] **by** *auto*

thm *ptran'-def*

lemma *cpts-p-simpsI*:
$((\exists P\ s.\ aa = [(P, s)]) \vee$
$(\exists P\ t\ xs\ s.\ aa = (P, s)\ \#\ (P, t)\ \#\ xs \wedge (P, t)\ \#\ xs \in$ *cpts-pI* $\Gamma) \vee$
$(\exists P\ s\ Q\ t\ xs.\ aa = (P, s)\ \#\ (Q, t)\ \#\ xs \wedge ptran'\ \Gamma\ (P, s)\ (Q, t) \wedge (Q, t)\ \#\ xs \in$ *cpts-pI* $\Gamma))$
$\Longrightarrow (aa \in$ *cpts-pI* $\Gamma)$

apply(*simp add*: *ptran′-def*) **using** *cptn.simps*[*of aa*] **by** *blast*

lemma *cpts-of-p-defI*: *l*!*0*=(*P*,*s*) ∧ *l* ∈ *cpts-pI* Γ ⟹ *l* ∈ *cpts-of-pI* Γ *P s*
by(*simp add*: *cp-def*)

interpretation *event-comp ptranI petranI None cpts-pI cpts-of-pI*
using *cpts-pI-ne-empty cpts-p-simpsI cpts-of-p-defI petran-simpsI none-no-tranI ptran-neqI*
event-comp.intro[*of ptranI petranI None cpts-pI cpts-of-pI*] *event.intro*[*of petranI None ptranI*]
event-comp-axioms.intro[*of cpts-pI ptranI cpts-of-pI*]
apply(*simp add*: *ptran′-def*) **by** *blast*

lemma *prog-validity-defI*: *prog-validityI* Γ *P pre rely guar post* ⟹
∀ *s*. *cpts-of-pI* Γ *P s* ∩ *assume-pI* Γ (*pre*, *rely*) ⊆ *commit-pI* Γ (*guar*, *post*)
by (*simp add*: *prog-validity-def*)

lemma *assume-p-defI*: *snd* (*c*!*0*) ∈ *pre* ∧ (∀ *i*. *Suc i*<*length c* ⟶
petranI Γ (*c*!*i*) (*c*!(*Suc i*)) ⟶ (*snd* (*c*!*i*), *snd* (*c*!*Suc i*)) ∈ *rely*) ⟹
c ∈ *assume-pI* Γ (*pre*, *rely*)
by(*simp add*: *assum-def PiCore-Semantics.gets-p-def*)

lemma *commit-p-defI*: *c* ∈ *commit-pI* Γ (*guar*, *post*) ⟹ (∀ *i*. *Suc i*<*length c* ⟶

(*c*!*i*,*c*!(*Suc i*)) ∈ *ptranI* Γ ⟶ (*snd* (*c*!*i*), *snd* (*c*!*Suc i*)) ∈ *guar*) ∧
(*fst* (*last c*) = *None* ⟶ *snd* (*last c*) ∈ *post*)
by(*simp add*: *comm-def PiCore-Semantics.getspc-p-def*
PiCore-Semantics.gets-p-def)

lemma *rgsound-pI*: *rghoare-pI* Γ *P pre rely guar post* ⟶ *prog-validityI* Γ *P pre rely guar post*
using *rgsound-p* **by** *blast*

interpretation *event-hoare ptranI petranI None cpts-pI cpts-of-pI prog-validityI assume-pI commit-pI rghoare-pI*
using *cpts-pI-ne-empty cpts-p-simpsI cpts-of-p-defI petran-simpsI none-no-tranI ptran-neqI*
prog-validity-defI assume-p-defI commit-p-defI rgsound-pI
event-comp-axioms.intro[*of cpts-pI ptranI cpts-of-pI*]
event-comp.intro[*of ptranI petranI None cpts-pI cpts-of-pI*] *event.intro*[*of petranI None ptranI*]
event-validity-axioms.intro[*of prog-validityI cpts-of-pI assume-pI commit-pI petranI ptranI None*]
event-validity.intro[*of ptranI petranI None cpts-pI cpts-of-pI prog-validityI assume-pI commit-pI*]
event-hoare.intro[*of ptranI petranI None cpts-pI cpts-of-pI prog-validityI assume-pI commit-pI rghoare-pI*]

event-hoare-axioms.intro[*of rghoare-pI prog-validityI*]
apply(*simp add*: *ptran′-def gets-p-def getspc-p-def*) **by** *blast*

thm *invariant-theorem*

lemma *stable-eq*[*simp*]: *stable-e* = *stable*
by(*simp add*:*stable-e-def stable-def*)

end

3 Concrete Syntax of PiCore-SIMP

theory *picore-SIMP-Syntax*
imports *picore-SIMP*

begin

syntax
-quote :: ′*b* ⇒ (′*s* ⇒ ′*b*) ((≪-≫) [*0*] *1000*)
-antiquote :: (′*s* ⇒ ′*b*) ⇒ ′*b* (´- [*1000*] *1000*)
-Assert :: ′*s* ⇒ ′*s set* (({⦃-⦄}) [*0*] *1000*)

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

parse-translation ‹
let
fun quote-tr [*t*] = *Syntax-Trans.quote-tr* @{*syntax-const -antiquote*} *t*
| *quote-tr ts* = *raise TERM* (*quote-tr*, *ts*);
in [(@{*syntax-const -quote*}, *K quote-tr*)] *end*
›

definition *Skip* :: ′*s com* (*SKIP*)
where *SKIP* ≡ *Basic id*

abbreviation *Wrap-prog* :: ′*s com* ⇒ ′*s com option* (*W*(-) *0*)
where *Wrap-prog p* ≡ *Some p*

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

syntax
rghoare-p :: ′*Env* ⇒ ′*prog* ⇒ [′*s set*, (′*s* × ′*s*) *set*, (′*s* × ′*s*) *set*, ′*s set*] ⇒ *bool*
(- ⊢ - sat_p [-, -, -, -] [*60*,*60*,*0*,*0*,*0*,*0*] *45*)
rghoare-e :: ′*Env* ⇒ (′*l*,′*k*,′*s*,′*prog*) *event* ⇒ [′*s set*, (′*s* × ′*s*) *set*, (′*s* × ′*s*) *set*, ′*s set*] ⇒ *bool*

$(- \vdash - \ sat_e \ [-, -, -, -] \ [60,60,0,0,0,0] \ 45)$
Evt-sat-RG:: $'Env \Rightarrow ('l,'k,'s,'prog)$ *event* $\Rightarrow$ $'s$ *rgformula* $\Rightarrow$ *bool* $((- \dashv\vdash -) \ [60,60,60] \ 61)$
rghoare-es :: $'Env \Rightarrow ('l,'k,'s,'prog)$ *rgformula-ess* $\Rightarrow [{'s}\ set, ({'s} \times {'s})\ set, ({'s} \times {'s})\ set, {'s}\ set] \Rightarrow bool$
$(- \vdash - \ sat_s \ [-, -, -, -] \ [60,60,0,0,0,0] \ 45)$
rghoare-pes :: $'Env \Rightarrow ('l,'k,'s,'prog)$ *rgformula-par* $\Rightarrow [{'s}\ set, ({'s} \times {'s})\ set, ({'s} \times {'s})\ set, {'s}\ set] \Rightarrow bool$
$(- \vdash - \ SAT \ [-, -, -, -] \ [60,60,0,0,0,0] \ 45)$
Evt-sat-RG:: $'Env \Rightarrow ('l,'k,'s,'prog)$ *event* $\Rightarrow$ $'s$ *rgformula* $\Rightarrow$ *bool* $((- \dashv\vdash -) \ [60,60,60] \ 61)$
Esys-sat-RG :: $'Env \Rightarrow ('l,'k,'s,'prog)$ *rgformula-ess* $\Rightarrow$ $'s$ *rgformula* $\Rightarrow$ *bool* $((- \dashv\vdash_{es} -) \ [60,60,60] \ 61)$

syntax
-Assign :: $idt \Rightarrow {'b} \Rightarrow {'s}\ com$ $((\acute{} - := / -) \ [70, 65] \ 61)$
-Cond :: $'s\ bexp \Rightarrow {'s}\ com \Rightarrow {'s}\ com \Rightarrow {'s}\ com$ ((0IF -/ THEN -/ ELSE -/FI) [0, 0, 0] 61)
-Cond2 :: $'s\ bexp \Rightarrow {'s}\ com \Rightarrow {'s}\ com$ ((0IF - THEN - FI) [0,0] 62)
-While :: $'s\ bexp \Rightarrow {'s}\ com \Rightarrow {'s}\ com$ ((0WHILE - /DO - /OD) [0, 0] 61)
-Await :: $'s\ bexp \Rightarrow {'s}\ com \Rightarrow {'s}\ com$ ((0AWAIT - /THEN /- /END) [0,0] 61)
-Atom :: $'s\ com \Rightarrow {'s}\ com$ ((0ATOMIC - END) 61)
-Wait :: $'s\ bexp \Rightarrow {'s}\ com$ ((0WAIT - END) 61)
-For :: $'s\ com \Rightarrow {'s}\ bexp \Rightarrow {'s}\ com \Rightarrow {'s}\ com \Rightarrow {'s}\ com$ ((0FOR -;/ -;/ -/ DO -/ ROF))
-Event :: $['a, 'a, 'a] \Rightarrow ('l,'k,'s,'s\ com\ option)\ event$ ((EVENT - WHEN - THEN - END) [0,0,0] 61)
-Event2 :: $['a, 'a] \Rightarrow ('l,'k,'s,'s\ com\ option)\ event$ ((EVENT - THEN - END) [0,0] 61)
-Event-a :: $['a, 'a, 'a] \Rightarrow ('l,'k,'s,'s\ com\ option)\ event$ (($EVENT_A$ - WHEN - THEN - END) [0,0,0] 61)

translations
$\acute{}x := a \rightharpoonup$ *CONST Basic* $\ll\acute{}$(*-update-name x* (λ-. *a*))$\gg$
IF b THEN c1 ELSE c2 FI $\rightharpoonup$ *CONST Cond* $\{\!|b|\!\}$ *c1 c2*
IF b THEN c FI $\rightleftharpoons$ *IF b THEN c ELSE SKIP FI*
WHILE b DO c OD $\rightharpoonup$ *CONST While* $\{\!|b|\!\}$ *c*
AWAIT b THEN c END $\rightleftharpoons$ *CONST Await* $\{\!|b|\!\}$ *c*

ATOMIC c END $\rightleftharpoons$ *AWAIT CONST True THEN c END*
WAIT b END $\rightleftharpoons$ *AWAIT b THEN SKIP END*
FOR a; b; c DO p ROF $\rightharpoonup$ *a;; WHILE b DO p;;c OD*
EVENT l WHEN g THEN bd END $\rightharpoonup$ *CONST BasicEvent* (*l*,($\{\!|g|\!\}$, *W*(*bd*)))
EVENT l THEN bd END $\rightleftharpoons$ *EVENT l WHEN CONST True THEN bd END*
$EVENT_A$ *l WHEN g THEN bd END* $\rightleftharpoons$ *EVENT l THEN* (*AWAIT g THEN bd END*) *END*

Translations for variables before and after a transition:

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

translations
  $^{o}x \rightleftharpoons x$ *´CONST fst*
  $^{a}x \rightleftharpoons x$ *´CONST snd*

print-translation ‹
```
  let
    fun quote-tr' f (t :: ts) =
          Term.list-comb (f $ Syntax-Trans.quote-tr' @{syntax-const -antiquote} t,
ts)
      | quote-tr' - - = raise Match;

    val assert-tr' = quote-tr' (Syntax.const @{syntax-const -Assert});

    fun bexp-tr' name ((Const (@{const-syntax Collect}, -) $ t) :: ts) =
          quote-tr' (Syntax.const name) (t :: ts)
      | bexp-tr' - - = raise Match;

    fun assign-tr' (Abs (x, -, f $ k $ Bound 0) :: ts) =
        quote-tr' (Syntax.const @{syntax-const -Assign} $ Syntax-Trans.update-name-tr'
f)
            (Abs (x, dummyT, Syntax-Trans.const-abs-tr' k) :: ts)
      | assign-tr' - = raise Match;
  in
   [(@{const-syntax Collect}, K assert-tr'),
    (@{const-syntax Basic}, K assign-tr'),
    (@{const-syntax Cond}, K (bexp-tr' @{syntax-const -Cond})),
    (@{const-syntax While}, K (bexp-tr' @{syntax-const -While}))]
  end
```
›

lemma *colltrue-eq-univ*[*simp*]: ⦃*True*⦄ = *UNIV* **by** *auto*

end

4 Lemmas of Picore-SIMP

theory *picore-SIMP-lemma*
imports *picore-SIMP-Syntax picore-SIMP*

begin

lemma *id-belong*[*simp*]: *Id* ⊆⦃$^{a}x = {}^{o}x$⦄
  by (*simp add*: *Collect-mono Id-fstsnd-eq*)

lemma *allpre-eq-pre*: $(\forall v \in U.\ \vdash_I P\ sat_p\ [\{v\},\ rely,\ guar,\ post]) \longleftrightarrow\ \vdash_I P\ sat_p$

$[U,\ rely,\ guar,\ post]$
  apply *auto* **using** *Allprecond* **apply** *blast*
  using *Conseq[of - - rely rely guar guar post post P]* **by** *auto*

lemma *sat-pre-imp-allinpre*: $\vdash_I P\ sat_p\ [U,\ rely,\ guar,\ post] \Longrightarrow v{\in}U \Longrightarrow \vdash_I P\ sat_p\ [\{v\},\ rely,\ guar,\ post]$
  using *Conseq[of - - rely rely guar guar post post P]* **by** *auto*

lemma *stable-int-col2*: $stable\ \{\!|s|\!\}\ r \Longrightarrow stable\ \{\!|t|\!\}\ r \Longrightarrow stable\ \{\!|s \wedge t|\!\}\ r$
  by *auto*

lemma *stable-int-col3*: $stable\ \{\!|k|\!\}\ r \Longrightarrow stable\ \{\!|s|\!\}\ r \Longrightarrow stable\ \{\!|t|\!\}\ r \Longrightarrow stable\ \{\!|k \wedge s \wedge t|\!\}\ r$
  by *auto*

lemma *stable-int-col4*: $stable\ \{\!|m|\!\}\ r \Longrightarrow stable\ \{\!|k|\!\}\ r \Longrightarrow stable\ \{\!|s|\!\}\ r$
  $\Longrightarrow stable\ \{\!|t|\!\}\ r \Longrightarrow stable\ \{\!|m \wedge k \wedge s \wedge t|\!\}\ r$
  by *auto*

lemma *stable-int-col5*: $stable\ \{\!|q|\!\}\ r \Longrightarrow stable\ \{\!|m|\!\}\ r \Longrightarrow stable\ \{\!|k|\!\}\ r$
  $\Longrightarrow stable\ \{\!|s|\!\}\ r \Longrightarrow stable\ \{\!|t|\!\}\ r \Longrightarrow stable\ \{\!|q \wedge m \wedge k \wedge s \wedge t|\!\}\ r$
  by *auto*

lemma *stable-un2*: $stable\ s\ r \Longrightarrow stable\ t\ r \Longrightarrow stable\ (s \cup t)\ r$
  by (*simp add*: *stable-def*)

lemma *stable-un-R*: $stable\ s\ r \Longrightarrow stable\ s\ r' \Longrightarrow stable\ s\ (r \cup r')$
  by (*meson UnE stable-def*)

lemma *stable-un-S*: $\forall\, t.\ stable\ s\ (P\ t) \Longrightarrow stable\ s\ (\bigcup t.\ P\ t)$
apply(*simp add*:*stable-def*) **by** *auto*

lemma *stable-un-S2*: $\forall\, t\ x.\ stable\ s\ (P\ t\ x) \Longrightarrow stable\ s\ (\bigcup t\ x.\ P\ t\ x)$
apply(*simp add*:*stable-def*) **by** *auto*

lemma *pairv-IntI*:
$y \in \{\!|\acute{}(Pair\ V) \in A|\!\} \Longrightarrow y \in \{\!|\acute{}(Pair\ V) \in B|\!\} \Longrightarrow y \in \{\!|\acute{}(Pair\ V) \in A \cap B|\!\}$
by *auto*

lemma *pairv-rId*:
$y \in \{\!|\acute{}(Pair\ V) \in A|\!\} \Longrightarrow y \in \{\!|\acute{}(Pair\ V) \in A \cup Id|\!\}$
by *auto*

end

5 Formal Specification and Reasoning of an Interruptable Controller for Stepper Motor

theory *IRQStepperMotor*
imports *../../Adapter-SIMP/picore-SIMP-lemma*
begin

5.1 functional specification

datatype *Device* = *Ctrl* | *Radar* | *PIC*

datatype *Irq* = *C* | *R*

record *State* = *stack* :: *Irq list*
iflag :: *bool*
car-pos :: *int*
obstacle-pos :: *int list*
i :: *int*
pos-aux :: *int*
obst-pos-aux :: *int list*

datatype *EL* = *ForwardH* | *BackwardH* | *ObstacleH* | *IRQsE*

print-theorems

datatype *Parameter* = *Irq Irq* | *Integer int* | *Str string* | *Natural nat*

type-synonym *EventLabel* = *EL* × (*Parameter list* × *Device*)

definition *get-evt-label* :: *EL* ⇒ *Parameter list* ⇒ *Device* ⇒ *EventLabel* (- - @ - [*0*,*0*,*0*] *20*)
where *get-evt-label el ps k* ≡ (*el*,(*ps*,*k*))

definition *iret* :: *State com*
where *iret* ≡ ´*stack* := *tl* ´*stack*

definition *push* :: *Irq* ⇒ *State com*
where *push d* ≡ ´*stack* := *d* # (´*stack*)

definition *cli* :: *State com*
where *cli* ≡ ´*iflag* := *False*

definition *sti* :: *State com*
where *sti* ≡ ´*iflag* := *True*

definition *stm* :: *Irq* ⇒ *State com* ⇒ *State com* (- ▶ -)
where *stm d p* ≡ *AWAIT hd* ´*stack* = *d THEN p END*

definition *willcollide* :: *int* ⇒ *int* ⇒ *int list* ⇒ *bool*
  where *willcollide s t l* ≡ *find* ($\lambda x.\ s \leq x \wedge x \leq t$) *l* = *None*

definition *collide* :: *'a* ⇒ *'a list* ⇒ *bool*
  where *collide pos l* ≡ *find* ($\lambda x.\ x = pos$) *l* ≠ *None*

definition *IRQs* :: *Irq* ⇒ (*EventLabel*, *Device*, *State*, *State com option*) *event*
  where *IRQs d* ≡
    EVENT IRQsE [*Irq d*] @ *PIC*
    THEN
      ATOMIC
        (∗*the interrupt is the one being handled have to be delayed*(*skipped*)
          the interrupt should not be the PIC ∗)
        IF hd ´stack ≠ *d THEN push d FI*
      END
    END

definition *forward* :: *nat* ⇒ (*EventLabel*, *Device*, *State*, *State com option*) *event*
  where *forward v* ≡
    EVENT ForwardH [*Natural v*] @ *Ctrl*
    THEN
      (*C* ▶ *´i* := *0*);;
      (*C* ▶ *´pos-aux* := *´car-pos*);;
      WHILE ´i ≠ *int v* ∧ ¬*collide* (*´car-pos* + *1*) *´obstacle-pos DO*
        (*C* ▶ *ATOMIC*
                IF ¬*collide* (*´car-pos* + *1*) *´obstacle-pos THEN*
                  ´car-pos := *´car-pos* + *1*
                FI
              END);;
        (*C* ▶ *´i* := *´i* + *1*)
      OD;;
      (*C* ▶ *iret*)
    END

definition *backward* :: *nat* ⇒ (*EventLabel*, *Device*, *State*, *State com option*) *event*

  where *backward v* ≡
    EVENT BackwardH [*Natural v*] @ *Ctrl*
    THEN
      (*C* ▶ *´i* := *0*);;
      (*C* ▶ *´pos-aux* := *´car-pos*);;
      WHILE ´i ≠ *int v* ∧ ¬*collide* (*´car-pos* − *1*) *´obstacle-pos DO*
        (*C* ▶ *ATOMIC*
                IF ¬*collide* (*´car-pos* − *1*) *´obstacle-pos THEN*
                  ´car-pos := *´car-pos* − *1*

```
          FI
         END);;
      (C ▶ ´i := ´i + 1)
    OD;;
    (C ▶ iret)
   END
```

definition *obstacle* :: *int* ⇒ (*EventLabel*, *Device*, *State*, *State com option*) *event*
  where *obstacle v* ≡

```
   EVENT ObstacleH [Integer v] @ Radar
   THEN
    (R ▶ ´obst-pos-aux := ´obstacle-pos);;
    (R ▶ IF v ≠ ´car-pos ∧ v ≠ ´car-pos + 1 ∧ v ≠ ´car-pos − 1 THEN
          ´obstacle-pos := v # ´obstacle-pos
       FI);;
    (R ▶ iret)
   END
```

5.2 Rely-guarantee condition of events

abbreviation *forward-rely* ≡ {|ª*car-pos* = º*car-pos* ∧ ª*i* = º*i* ∧ ª*pos-aux* = º*pos-aux*

∧ (*hd* º*stack* ≠ *C* ⟶ ((ª*stack* = *tl* º*stack* ∨ ª*obst-pos-aux* = º*obstacle-pos*

∨ ª*stack* = *C* # º*stack*) ∧ ª*obstacle-pos* = º*obstacle-pos*)
∨ (*set* º*obstacle-pos* ⊆ *set* ª*obstacle-pos*
∧ *collide* (º*car-pos* + *1*) º*obstacle-pos* = *collide*
(ª*car-pos* + *1*) ª*obstacle-pos*))
∧ (*hd* º*stack* = *C* ⟶ º*obstacle-pos* = ª*obstacle-pos* ∧ ª*stack* = *R* #
º*stack*
∧ º*obst-pos-aux* = ª*obst-pos-aux*)|} ∪ *Id*

abbreviation *forward-guar* ≡ {|*hd* º*stack* = *C* ∧ (((ª*i* = *0* ∨ ª*i* = º*i* + *1* ∨ ª*stack*
= *tl* º*stack*) ∧ ª*car-pos* = º*car-pos*) ∨
(¬*collide* (º*car-pos* + *1*) º*obstacle-pos* ∧ ª*car-pos* = º*car-pos* + *1*))
∧ ª*obstacle-pos* = º*obstacle-pos* ∧ ª*obst-pos-aux* = º*obst-pos-aux*|} ∪ *Id*

abbreviation *forward-post v* ≡ {| ´*car-pos* = ´*pos-aux* + ´*i* ∧
(´*i* = *int v* ∨ *collide* (´*pos-aux* + ´*i* + *1*) ´*obstacle-pos*) |}

definition *forward-RGCond* :: *nat* ⇒ (*State*) *PiCore-Hoare.rgformula*
  where *forward-RGCond v* ≡
    RG[{|*True*|}, *forward-rely*, *forward-guar*, *forward-post v*]

abbreviation *backward-rely* ≡ {|ª*car-pos* = º*car-pos* ∧ ª*i* = º*i* ∧ ª*pos-aux* =
º*pos-aux*
∧ (*hd* º*stack* ≠ *C* ⟶ ((ª*stack* = *tl* º*stack* ∨ ª*obst-pos-aux* = º*obstacle-pos*

$\vee$ ${}^{\mathrm{a}}stack = C \ \# \ {}^{\mathrm{o}}stack) \wedge {}^{\mathrm{a}}obstacle\text{-}pos = {}^{\mathrm{o}}obstacle\text{-}pos)$
$\vee$ $(set\ {}^{\mathrm{o}}obstacle\text{-}pos \subseteq set\ {}^{\mathrm{a}}obstacle\text{-}pos$
$\wedge$ $collide\ ({}^{\mathrm{o}}car\text{-}pos - 1)\ {}^{\mathrm{o}}obstacle\text{-}pos = collide$
$({}^{\mathrm{a}}car\text{-}pos - 1)\ {}^{\mathrm{a}}obstacle\text{-}pos))$
$\wedge$ $(hd\ {}^{\mathrm{o}}stack = C \longrightarrow {}^{\mathrm{o}}obstacle\text{-}pos = {}^{\mathrm{a}}obstacle\text{-}pos \wedge {}^{\mathrm{a}}stack = R \ \#$
${}^{\mathrm{o}}stack$
$\wedge$ ${}^{\mathrm{o}}obst\text{-}pos\text{-}aux = {}^{\mathrm{a}}obst\text{-}pos\text{-}aux)\} \cup Id$

abbreviation $backward\text{-}guar \equiv \{hd\ {}^{\mathrm{o}}stack = C \wedge ((({}^{\mathrm{a}}i = 0 \vee {}^{\mathrm{a}}i = {}^{\mathrm{o}}i + 1 \vee$
${}^{\mathrm{a}}stack = tl\ {}^{\mathrm{o}}stack) \wedge {}^{\mathrm{a}}car\text{-}pos = {}^{\mathrm{o}}car\text{-}pos) \vee$
$(\neg collide\ ({}^{\mathrm{o}}car\text{-}pos - 1)\ {}^{\mathrm{o}}obstacle\text{-}pos \wedge {}^{\mathrm{a}}car\text{-}pos = {}^{\mathrm{o}}car\text{-}pos - 1))$
$\wedge\ {}^{\mathrm{a}}obstacle\text{-}pos = {}^{\mathrm{o}}obstacle\text{-}pos \wedge {}^{\mathrm{a}}obst\text{-}pos\text{-}aux = {}^{\mathrm{o}}obst\text{-}pos\text{-}aux\} \cup Id$

abbreviation $backward\text{-}post\ v \equiv \{\acute{}car\text{-}pos = \acute{}pos\text{-}aux - \acute{}i\ \wedge$
$(\acute{}i = int\ v \vee collide\ (\acute{}pos\text{-}aux - \acute{}i - 1)\ \acute{}obstacle\text{-}pos)\}$

definition $backward\text{-}RGCond :: nat \Rightarrow (State)\ PiCore\text{-}Hoare.rgformula$
where $backward\text{-}RGCond\ v \equiv$
$RG[\{True\}, backward\text{-}rely, backward\text{-}guar, backward\text{-}post\ v]$

abbreviation $obstacle\text{-}rely \equiv \{{}^{\mathrm{a}}obstacle\text{-}pos = {}^{\mathrm{o}}obstacle\text{-}pos \wedge {}^{\mathrm{a}}obst\text{-}pos\text{-}aux =$
${}^{\mathrm{o}}obst\text{-}pos\text{-}aux\ \wedge$
$(hd\ {}^{\mathrm{o}}stack \neq R \longrightarrow {}^{\mathrm{a}}i = 0 \vee {}^{\mathrm{a}}i = {}^{\mathrm{o}}i + 1 \vee {}^{\mathrm{a}}stack = tl\ {}^{\mathrm{o}}stack$
$\vee\ (\neg collide\ ({}^{\mathrm{o}}car\text{-}pos + 1)\ {}^{\mathrm{o}}obstacle\text{-}pos \wedge {}^{\mathrm{a}}car\text{-}pos = {}^{\mathrm{o}}car\text{-}pos + 1)$
$\vee\ (\neg collide\ ({}^{\mathrm{o}}car\text{-}pos - 1)\ {}^{\mathrm{o}}obstacle\text{-}pos \wedge {}^{\mathrm{a}}car\text{-}pos = {}^{\mathrm{o}}car\text{-}pos - 1)$
$\vee\ {}^{\mathrm{a}}stack = R \ \# \ {}^{\mathrm{o}}stack) \wedge$
$(hd\ {}^{\mathrm{o}}stack = R \longrightarrow {}^{\mathrm{o}}car\text{-}pos = {}^{\mathrm{a}}car\text{-}pos \wedge {}^{\mathrm{o}}i = {}^{\mathrm{a}}i \wedge {}^{\mathrm{a}}pos\text{-}aux = {}^{\mathrm{o}}pos\text{-}aux$

$\wedge\ {}^{\mathrm{a}}stack = C \ \# \ {}^{\mathrm{o}}stack)\} \cup Id$

abbreviation $obstacle\text{-}guar \equiv$
$\{hd\ {}^{\mathrm{o}}stack = R \wedge ((({}^{\mathrm{a}}stack = tl\ {}^{\mathrm{o}}stack \vee {}^{\mathrm{a}}obst\text{-}pos\text{-}aux = {}^{\mathrm{o}}obstacle\text{-}pos) \wedge$
${}^{\mathrm{a}}obstacle\text{-}pos = {}^{\mathrm{o}}obstacle\text{-}pos)$
$\vee\ (set\ {}^{\mathrm{o}}obstacle\text{-}pos \subseteq set\ {}^{\mathrm{a}}obstacle\text{-}pos$
$\wedge\ collide\ ({}^{\mathrm{o}}car\text{-}pos - 1)\ {}^{\mathrm{o}}obstacle\text{-}pos = collide$
$({}^{\mathrm{a}}car\text{-}pos - 1)\ {}^{\mathrm{a}}obstacle\text{-}pos$
$\wedge\ collide\ {}^{\mathrm{o}}car\text{-}pos\ {}^{\mathrm{o}}obstacle\text{-}pos = collide\ {}^{\mathrm{a}}car\text{-}pos$
${}^{\mathrm{a}}obstacle\text{-}pos$
$\wedge\ collide\ ({}^{\mathrm{o}}car\text{-}pos + 1)\ {}^{\mathrm{o}}obstacle\text{-}pos = collide$
$({}^{\mathrm{a}}car\text{-}pos + 1)\ {}^{\mathrm{a}}obstacle\text{-}pos))$
$\wedge\ {}^{\mathrm{a}}car\text{-}pos = {}^{\mathrm{o}}car\text{-}pos \wedge {}^{\mathrm{a}}i = {}^{\mathrm{o}}i \wedge {}^{\mathrm{a}}pos\text{-}aux = {}^{\mathrm{o}}pos\text{-}aux\} \cup Id$

abbreviation $obstacle\text{-}post\ v \equiv \{\acute{}obstacle\text{-}pos = v \ \# \ \acute{}obst\text{-}pos\text{-}aux \vee \acute{}obstacle\text{-}pos$
$= \acute{}obst\text{-}pos\text{-}aux\}$

definition $obstacle\text{-}RGCond :: int \Rightarrow (State)\ PiCore\text{-}Hoare.rgformula$
where $obstacle\text{-}RGCond\ v \equiv$
$RG[\{True\}, obstacle\text{-}rely, obstacle\text{-}guar, obstacle\text{-}post\ v]$

abbreviation *IRQs-guar d* $\equiv$
$\{\!|hd\ ^{\mathrm{o}}stack \neq d \wedge {}^{\mathrm{a}}stack = d \# {}^{\mathrm{o}}stack \wedge {}^{\mathrm{a}}car\text{-}pos = {}^{\mathrm{o}}car\text{-}pos$
$\wedge {}^{\mathrm{a}}i = {}^{\mathrm{o}}i \wedge {}^{\mathrm{a}}pos\text{-}aux = {}^{\mathrm{o}}pos\text{-}aux \wedge {}^{\mathrm{a}}obstacle\text{-}pos = {}^{\mathrm{o}}obstacle\text{-}pos$
$\wedge {}^{\mathrm{a}}obst\text{-}pos\text{-}aux = {}^{\mathrm{o}}obst\text{-}pos\text{-}aux|\!\} \cup Id$

definition *IRQs-RGCond* :: *Irq* $\Rightarrow$ (*State*) *PiCore-Hoare.rgformula*
where *IRQs-RGCond d* $\equiv$
RG[$\{\!|True|\!\}$, $\{\!|True|\!\}$, *IRQs-guar d*, $\{\!|True|\!\}$]

definition *forward-RGF* :: *nat* $\Rightarrow$ (*EventLabel*, *Device*, *State*, *State com option*) *rgformula-e*
where *forward-RGF v* $\equiv$ (*forward v*, *forward-RGCond v*)

definition *backward-RGF* :: *nat* $\Rightarrow$ (*EventLabel*, *Device*, *State*, *State com option*) *rgformula-e*
where *backward-RGF v* $\equiv$ (*backward v*, *backward-RGCond v*)

definition *obstacle-RGF* :: *int* $\Rightarrow$ (*EventLabel*, *Device*, *State*, *State com option*) *rgformula-e*
where *obstacle-RGF v* $\equiv$ (*obstacle v*, *obstacle-RGCond v*)

definition *IRQs-RGF* :: *Irq* $\Rightarrow$ (*EventLabel*, *Device*, *State*, *State com option*) *rgformula-e*
where *IRQs-RGF r* $\equiv$ (*IRQs r*, *IRQs-RGCond r*)

definition *EvtSys-on-Motor-RGF* :: (*EventLabel*, *Device*, *State*, *State com option*) *rgformula-es*
where *EvtSys-on-Motor-RGF* $\equiv$
(*rgf-EvtSys* (($\bigcup v.\{forward\text{-}RGF\ v\}$) $\cup$
($\bigcup v.\ \{backward\text{-}RGF\ v\}$)),
RG[$\{\!|True|\!\}$, (*forward-rely* $\cap$ *backward-rely*), (*forward-guar* $\cup$ *backward-guar*),
($\bigcup v.\ forward\text{-}post\ v \cup backward\text{-}post\ v$)])

definition *EvtSys-on-Radar-RGF* :: (*EventLabel*, *Device*, *State*, *State com option*) *rgformula-es*
where *EvtSys-on-Radar-RGF* $\equiv$
(*rgf-EvtSys* ($\bigcup v.\{obstacle\text{-}RGF\ v\}$),
RG[$\{\!|True|\!\}$, *obstacle-rely*, *obstacle-guar*, ($\bigcup v.\ obstacle\text{-}post\ v$)])

definition *EvtSys-on-PIC-RGF* :: (*EventLabel*, *Device*, *State*, *State com option*) *rgformula-es*
where *EvtSys-on-PIC-RGF* $\equiv$
(*rgf-EvtSys* ($\bigcup d.\ \{IRQs\text{-}RGF\ d\}$),
RG[$\{\!|True|\!\}$, $\{\!|True|\!\}$, ($\bigcup d.\ IRQs\text{-}guar\ d$), $\{\!|True|\!\}$])

definition *Carsystem-Spec* :: (*EventLabel*, *Device*, *State*, *State com option*) *rgformula-par*
where *Carsystem-Spec k* ≡ *case k of Ctrl* ⇒ *EvtSys-on-Motor-RGF*
| *Radar* ⇒ *EvtSys-on-Radar-RGF*
| *PIC* ⇒ *EvtSys-on-PIC-RGF*

definition *init* :: *State*
where *init* ≡ ⦇*stack* = [], *iflag* = *True*, *car-pos* = *0*,
obstacle-pos = [], *i* = *0*,
pos-aux = *0*, *obst-pos-aux* = []⦈

consts *s0*::*State*
definition *s0-witness*::*State*
where *s0-witness* ≡ *init*

specification (*s0*)
s0-init: *s0* ≡ *init*
by *simp*

5.3 Functional correctness by rely guarantee proof

lemma *collide-subset*: *set a* ⊆ *set b* ⟹ *collide x a* ⟹ *collide x b*
unfolding *collide-def* **by** (*simp add*: *find-None-iff subset-eq*)

lemma *forward-satRG*: Γ (*forward v*) ⊢ *forward-RGCond v*
apply(*simp add*:*Evt-sat-RG-def*)
apply (*simp add*: *forward-def forward-RGCond-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def*
$Rely_f$*-def* $Guar_f$*-def getrgformula-def*)

apply(*rule Seq*[**where** *mid*=⦃´*car-pos* = ´*pos-aux* + ´*i* ∧ (*int v* = ´*i* ∨ *collide* (´*car-pos* + *1*) ´*obstacle-pos*) ⦄])
apply(*rule Seq*[**where** *mid*=⦃´*car-pos* = ´*pos-aux* + ´*i*⦄])
apply(*rule Seq*[**where** *mid*=⦃´*i* = *0*⦄])

apply(*simp add*:*stm-def*)
apply(*rule Await*)
apply(*simp add*:*stable-def*)+
apply(*rule allI*)
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+ **apply** *auto*[*1*]

apply(*simp add*:*stm-def*)
apply(*rule Await*)
apply(*simp add*:*stable-def*)+
apply(*rule allI*)

```
      apply(rule Basic)
        apply auto[1]
        apply(simp add:stable-def)+ apply auto[1]

    apply(rule While)
      apply(simp add:stable-def)
      apply(simp add: collide-def) apply auto[1]
        apply(simp add:stable-def) apply(rule allI) apply(rule impI)+ ap-
ply(rule allI)
        apply(case-tac int v = i x)
          apply auto[1]
          apply simp apply (metis collide-subset)

      apply(rule Seq[where mid={|´car-pos = ´pos-aux + ´i + 1|}])
        apply(simp add:stm-def)
        apply(rule Await)
          apply(simp add:stable-def) apply metis
          apply(simp add:stable-def)
            apply(rule allI)
            apply(rule Await)
              apply(simp add:stable-def) apply auto[1]
              apply(simp add:stable-def) apply auto[1]
              apply(rule allI)
              apply(rule Cond)
                apply(simp add:stable-def) apply auto[1]
                  apply(case-tac V = Va)
                    apply simp
                    apply(rule Basic)
                      apply auto[1]
                      apply(simp add:stable-def)
                      apply(simp add:stable-def) apply auto[1]
                      apply(simp add:stable-def) apply auto[1]
                    apply simp
                    apply(rule Basic)
                      apply simp+
                      apply(simp add:stable-def)+ apply auto[1]
                apply (simp add:Skip-def)
                apply(rule Basic)
                  apply auto[1]
                  apply simp
                  apply(simp add:stable-def) apply auto[1]
                  apply(simp add:stable-def) apply auto[1]
                  apply simp
        apply(simp add:stm-def)
        apply(rule Await)
          apply(simp add:stable-def)+
          apply(rule allI)
          apply(rule Basic)
            apply auto[1]
```

```
          apply(simp add:stable-def)+ apply auto[1]
        apply simp
        apply(simp add:stm-def)
        apply(rule Await)
          apply(simp add:stable-def) apply(rule allI) apply(rule impI)+
            apply(case-tac int v = i x)
              apply simp apply (metis collide-subset)

          apply(simp add:stable-def) apply(rule allI) apply(rule impI)+
            apply(case-tac int v = i x)
              apply simp
              apply simp apply (metis collide-subset)
          apply(rule allI)
          apply(simp add:iret-def)
          apply(rule Basic)
            apply auto[1]
            apply(simp add:stable-def)+ apply auto[1]
            apply(simp add:stable-def) apply auto[1]
    apply(simp add: stable-def Pre_f-def getrgformula-def Rely_f-def)
    apply(simp add: Guar_f-def getrgformula-def)
  done

lemma backward-satRG: Γ (backward v) ⊢ backward-RGCond v
  apply(simp add:Evt-sat-RG-def)
  apply (simp add: backward-def backward-RGCond-def)
    apply(rule BasicEvt)
      apply(simp add:body-def Pre_f-def Post_f-def guard-def
              Rely_f-def Guar_f-def getrgformula-def)

      apply(rule Seq[where mid={|´car-pos = ´pos-aux − ´i ∧ (int v = ´i ∨
collide (´car-pos − 1) ´obstacle-pos) |}])
      apply(rule Seq[where mid={|´car-pos = ´pos-aux − ´i|}])
       apply(rule Seq[where mid={|´i = 0|}])

        apply(simp add:stm-def)
        apply(rule Await)
          apply(simp add:stable-def)+
          apply(rule allI)
          apply(rule Basic)
            apply auto[1]
            apply(simp add:stable-def)+  apply auto[1]

        apply(simp add:stm-def)
        apply(rule Await)
          apply(simp add:stable-def)+
          apply(rule allI)
          apply(rule Basic)
            apply auto[1]
            apply(simp add:stable-def)+  apply auto[1]
```

apply(*rule While*)
apply(*simp add*:*stable-def*)
apply(*simp add*: *collide-def*) **apply** *auto*[*1*]
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*)+ **apply**(*rule allI*)
apply(*case-tac int v* = *i x*)
apply *auto*[*1*]
apply *simp* **apply** (*metis collide-subset*)

apply(*rule Seq*[**where** *mid*=⦃´*car-pos* = ´*pos-aux* − ´*i* − *1*⦄])
apply(*simp add*:*stm-def*)
apply(*rule Await*)
apply(*simp add*:*stable-def*) **apply** *metis*
apply(*simp add*:*stable-def*)
apply(*rule allI*)
apply(*rule Await*)
apply(*simp add*:*stable-def*) **apply** *auto*[*1*]
apply(*simp add*:*stable-def*) **apply** *auto*[*1*]
apply(*rule allI*)
apply(*rule Cond*)
apply(*simp add*:*stable-def*) **apply** *auto*[*1*]
apply(*case-tac V* = *Va*)
apply *simp*
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)
apply(*simp add*:*stable-def*) **apply** *auto*[*1*]
apply(*simp add*:*stable-def*) **apply** *auto*[*1*]
apply *simp*
apply(*rule Basic*)
apply *simp*+
apply(*simp add*:*stable-def*)+ **apply** *auto*[*1*]
apply (*simp add*:*Skip-def*)
apply(*rule Basic*)
apply *auto*[*1*]
apply *simp*
apply(*simp add*:*stable-def*) **apply** *auto*[*1*]
apply(*simp add*:*stable-def*) **apply** *auto*[*1*]
apply *simp*
apply(*simp add*:*stm-def*)
apply(*rule Await*)
apply(*simp add*:*stable-def*)+
apply(*rule allI*)
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+ **apply** *auto*[*1*]
apply *simp*
apply(*simp add*:*stm-def*)

```
        apply(rule Await)
          apply(simp add:stable-def) apply(rule allI) apply(rule impI)+
            apply(case-tac int v = i x)
              apply simp apply (metis collide-subset)

          apply(simp add:stable-def) apply(rule allI) apply(rule impI)+
            apply(case-tac int v = i x)
              apply simp
              apply simp apply (metis collide-subset)
          apply(rule allI)
          apply(simp add:iret-def)
          apply(rule Basic)
            apply auto[1]
            apply(simp add:stable-def)+ apply auto[1]
            apply(simp add:stable-def) apply auto[1]
    apply(simp add: stable-def Pre_f-def getrgformula-def Rely_f-def)
    apply(simp add: Guar_f-def getrgformula-def)
  done

lemma obstacle-satRG: Γ (obstacle v) ⊢ obstacle-RGCond v
  apply(simp add:Evt-sat-RG-def)
  apply (simp add: obstacle-def obstacle-RGCond-def)
  apply(rule BasicEvt)
    apply(simp add:body-def Pre_f-def Post_f-def guard-def
              Rely_f-def Guar_f-def getrgformula-def)
   apply(rule Seq[where mid={|´obstacle-pos = v # ´obst-pos-aux ∨ ´obstacle-pos
= ´obst-pos-aux|}])
     apply(rule Seq[where mid={|´obst-pos-aux = ´obstacle-pos|}])

       apply(simp add:stm-def)
       apply(rule Await)
         apply(simp add:stable-def)+
         apply(rule allI)
         apply(case-tac hd (stack V) = R)
           apply simp
           apply(rule Basic)
             apply simp+
             apply(simp add:stable-def)+ apply auto[1]
           apply simp
           apply(rule Basic)
             apply(simp add:stable-def)+

       apply(simp add:stm-def)
       apply(rule Await)
         apply(simp add:stable-def)+
         apply(rule allI)
         apply(rule Cond)
           apply(simp add:stable-def)
           apply(case-tac obst-pos-aux V = obstacle-pos V ∧ hd (stack V) = R
```

```
∧ v ≠ car-pos V ∧
                                        v ≠ car-pos V + 1 ∧
                                        v ≠ car-pos V −1)
            apply simp
            apply(rule Basic)
              apply(simp add:collide-def)
              apply auto[1]
              apply auto[1]
              apply(simp add:stable-def)+
            apply(rule Basic)
              apply(simp add:collide-def)
              apply auto[1]
              apply auto[1]
              apply(simp add:stable-def)+

          apply(simp add:Skip-def)
          apply(rule Basic)
            apply auto[1]
            apply(simp add:stable-def)+
        apply(simp add:stm-def)
          apply(rule Await)
          apply(simp add:stable-def)+
          apply(rule allI)
          apply(simp add:iret-def)
          apply(rule Basic)
            apply auto[1]
            apply simp
            apply(simp add:stable-def)+
    apply(simp add: stable-def Pre_f-def getrgformula-def Rely_f-def)
    apply(simp add: Guar_f-def getrgformula-def)
  done

lemma Interrupt-satRG: Γ (IRQs d) ⊢ IRQs-RGCond d
  apply(simp add:Evt-sat-RG-def)
  apply (simp add: IRQs-def IRQs-RGCond-def)
  apply(rule BasicEvt)
    apply(simp add:body-def Pre_f-def Post_f-def guard-def
              Rely_f-def Guar_f-def getrgformula-def)
    apply(rule Await)
      apply(simp add:stable-def)+
      apply(rule allI)
      apply(rule Cond)
        apply(simp add:stable-def)
        apply(simp add:push-def)
        apply(rule Basic)
          apply auto[1]
          apply(simp add:stable-def)+

        apply(simp add:Skip-def)
```

apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+

apply(*simp add*:*stable-def* Pre_f*-def* $Rely_f$*-def getrgformula-def*)
by(*simp add*:$Guar_f$*-def getrgformula-def*)

lemma *EvtSys-on-Motor-SatRG*:
$\Gamma \vdash$ *fst* (*EvtSys-on-Motor-RGF*) sat_s
[Pre_f (*snd* (*EvtSys-on-Motor-RGF*)),
$Rely_f$ (*snd* (*EvtSys-on-Motor-RGF*)),
$Guar_f$ (*snd* (*EvtSys-on-Motor-RGF*)),
$Post_f$ (*snd* (*EvtSys-on-Motor-RGF*))]
apply(*simp add*:*EvtSys-on-Motor-RGF-def* Pre_f*-def* $Rely_f$*-def*
$Guar_f$*-def* $Post_f$*-def getrgformula-def*)
apply(*rule EvtSys-h*)

apply *clarify*
apply(*case-tac* (*a*,*b*)$\in$ ($\bigcup v$. {*forward-RGF v*}))
using *forward-satRG forward-RGF-def Evt-sat-RG-def* E_e*-def* Pre_e*-def* $Rely_e$*-def*
$Guar_e$*-def* $Post_e$*-def*
$Guar_f$*-def* $Post_f$*-def* Pre_f*-def* $Rely_f$*-def snd-conv fst-conv UN-E singletonD*
apply *smt*
apply(*case-tac* (*a*,*b*)$\in$ ($\bigcup v$. {*backward-RGF v*}))
using *backward-satRG backward-RGF-def Evt-sat-RG-def* E_e*-def* Pre_e*-def* $Rely_e$*-def*
$Guar_e$*-def* $Post_e$*-def*
$Guar_f$*-def* $Post_f$*-def* Pre_f*-def* $Rely_f$*-def snd-conv fst-conv UN-E singletonD*
apply *smt*
apply *blast*

apply *clarify*
apply(*case-tac* (*a*,*b*)$\in$ ($\bigcup v$. {*forward-RGF v*}))
apply(*simp add*: *forward-RGF-def* E_e*-def* Pre_e*-def forward-RGCond-def getrgformula-def*)
apply *fastforce*
apply(*case-tac* (*a*,*b*)$\in$ ($\bigcup v$. {*backward-RGF v*}))
apply(*simp add*: *backward-RGF-def* E_e*-def* Pre_e*-def backward-RGCond-def*
getrgformula-def)
apply *fastforce*
apply *blast*

unfolding *Ball-def* **apply**(*rule allI*) **apply**(*rule impI*)
apply(*case-tac x*$\in$ ($\bigcup v$. {*forward-RGF v*}))
apply (*simp add*:*forward-RGF-def forward-RGCond-def* $Rely_e$*-def getrgformula-def*)

apply (*erule exE*) **apply** *auto*[*1*]
apply (*simp add*:*backward-RGF-def backward-RGCond-def* $Rely_e$*-def getrgformula-def*)

apply (*erule exE*) **apply** *auto*[*1*]

apply(*rule allI*) **apply**(*rule impI*)
apply(*case-tac* $x \in (\bigcup v.\ \{$*forward-RGF v*$\})$)
apply (*simp add*:*forward-RGF-def forward-RGCond-def* $Guar_e$*-def getrgformula-def*)

apply (*erule exE*) **apply** *auto*[*1*]
apply (*simp add*:*backward-RGF-def backward-RGCond-def* $Guar_e$*-def getrgformula-def*)

apply (*erule exE*) **apply** *auto*[*1*]

apply(*rule allI*) **apply**(*rule impI*)
apply(*case-tac* $x \in (\bigcup v.\ \{$*forward-RGF v*$\})$)
apply (*simp add*:*forward-RGF-def forward-RGCond-def* $Post_e$*-def getrgformula-def*)

apply (*erule exE*) **apply** *auto*[*1*]
apply (*simp add*:*backward-RGF-def backward-RGCond-def* $Post_e$*-def getrgformula-def*)

apply (*erule exE*) **apply** *auto*[*1*]

apply *auto*[*1*]
apply (*simp add*:*forward-RGF-def forward-RGCond-def backward-RGF-def backward-RGCond-def* Pre_e*-def* $Post_e$*-def getrgformula-def*)+

apply(*simp add*:*stable-def*)
by *simp*

lemma *EvtSys-on-Radar-SatRG*:
$\Gamma \vdash$ *fst* (*EvtSys-on-Radar-RGF*) sat_s
[Pre_f (*snd* (*EvtSys-on-Radar-RGF*)),
$Rely_f$ (*snd* (*EvtSys-on-Radar-RGF*)),
$Guar_f$ (*snd* (*EvtSys-on-Radar-RGF*)),
$Post_f$ (*snd* (*EvtSys-on-Radar-RGF*))]
apply(*simp add*:*EvtSys-on-Radar-RGF-def* Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def getrgformula-def*)
apply(*rule EvtSys-h*)

apply *auto*[*1*]
apply(*simp add*:E_e*-def obstacle-RGF-def*)
using *obstacle-satRG*
apply (*simp add*: *Evt-sat-RG-def* $Guar_e$*-def* $Guar_f$*-def* $Post_e$*-def* $Post_f$*-def* Pre_e*-def* Pre_f*-def* $Rely_e$*-def* $Rely_f$*-def*)

apply(*simp add*:Pre_e*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def*)
apply *fastforce*
apply(*simp add*: $Rely_e$*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def* Pre_e*-def*)
apply(*simp add*: $Guar_e$*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def* $Rely_e$*-def*)
apply(*simp add*: $Post_e$*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def* $Guar_e$*-def*)

apply(*simp add*: $Post_e$*-def* Pre_e*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def*)
apply *fastforce*
apply(*simp add*: $Post_e$*-def* Pre_e*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def*)
apply *fastforce*
apply(*simp add*:*stable-def*)
by *simp*

lemma *EvtSys-on-PIC-SatRG*:
$\Gamma \vdash$ *fst* (*EvtSys-on-PIC-RGF*) sat_s
[Pre_f (*snd* (*EvtSys-on-PIC-RGF*)),
$Rely_f$ (*snd* (*EvtSys-on-PIC-RGF*)),
$Guar_f$ (*snd* (*EvtSys-on-PIC-RGF*)),
$Post_f$ (*snd* (*EvtSys-on-PIC-RGF*))]
apply(*simp add*:*EvtSys-on-PIC-RGF-def* Pre_f*-def* $Rely_f$*-def*
$Guar_f$*-def* $Post_f$*-def getrgformula-def*)
apply(*rule EvtSys-h*)

apply *auto*[*1*]
apply(*simp add*:E_e*-def IRQs-RGF-def*)
using *Interrupt-satRG*
apply (*simp add*: *Evt-sat-RG-def* $Guar_e$*-def* $Guar_f$*-def* $Post_e$*-def* $Post_f$*-def*
Pre_e*-def* Pre_f*-def* $Rely_e$*-def* $Rely_f$*-def*)

apply(*simp add*:Pre_e*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*)
apply *fastforce*
apply(*simp add*: $Rely_e$*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*
Pre_e*-def*)
apply(*simp add*: $Guar_e$*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*
$Rely_e$*-def*)
apply(*simp add*: $Guar_e$*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*
$Rely_e$*-def*) **apply** *auto*[*1*]
apply(*simp add*: $Post_e$*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*)
apply *auto*[*1*]
apply(*simp add*: $Post_e$*-def* Pre_e*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*)

apply(*simp add*:*stable-def*)
by *simp*

definition *sys-guar* $\equiv$ (*forward-guar* $\cup$ *backward-guar*) $\cup$ *obstacle-guar* $\cup$ ($\bigcup d.$
IRQs-guar d)

lemma *esys-guar-in-sys*: $Guar_{es}$ (*Carsystem-Spec k*) $\subseteq$ *sys-guar*
apply(*induct k*)
apply(*simp add*:$Guar_{es}$*-def Carsystem-Spec-def EvtSys-on-Motor-RGF-def getrgformula-def*
sys-guar-def)
apply *fast*
apply(*simp add*:$Guar_{es}$*-def Carsystem-Spec-def EvtSys-on-Radar-RGF-def getrgformula-def*
sys-guar-def)

apply *fast*
apply(*simp add*:$Guar_{es}$*-def Carsystem-Spec-def EvtSys-on-PIC-RGF-def getrgformula-def sys-guar-def*)
done

lemma *esys-sat*: $\Gamma \vdash$ *fst* (*Carsystem-Spec k*)
sat$_s$ [Pre_{es} (*Carsystem-Spec k*),
$Rely_{es}$ (*Carsystem-Spec k*),
$Guar_{es}$ (*Carsystem-Spec k*),
$Post_{es}$ (*Carsystem-Spec k*)]
apply(*induct k*)
apply(*simp add*:*Carsystem-Spec-def* Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def*

getrgformula-def)
using *EvtSys-on-Motor-SatRG* **apply**(*simp add*:Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def*) **apply** *fast*

apply(*simp add*:*Carsystem-Spec-def* Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def*

getrgformula-def)
using *EvtSys-on-Radar-SatRG* **apply**(*simp add*:Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def*) **apply** *fast*

apply(*simp add*:*Carsystem-Spec-def* Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def*

getrgformula-def)
using *EvtSys-on-PIC-SatRG* **apply**(*simp add*:Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def*) **apply** *fast*
done

lemma *functional-correctness*: $\Gamma \vdash$ *Carsystem-Spec SAT* [{*s0*}, {}, *sys-guar*, {|*True*|}]
apply (*rule ParallelESys*)
apply(*simp add*:*Carsystem-Spec-def*)
apply(*rule allI*)
using *EvtSys-on-Motor-SatRG EvtSys-on-Radar-SatRG EvtSys-on-PIC-SatRG*
apply (*simp add*: $Guar_{es}$*-def* $Guar_f$*-def* $Post_{es}$*-def* $Post_f$*-def* Pre_{es}*-def* Pre_f*-def* $Rely_{es}$*-def* $Rely_f$*-def*)
apply (*smt Device.exhaust Device.simps*(*7*) *Device.simps*(*8*) *Device.simps*(*9*)
EvtSys-on-Motor-SatRG EvtSys-on-PIC-SatRG EvtSys-on-Radar-SatRG $Guar_f$*-def* $Post_f$*-def* Pre_f*-def* $Rely_f$*-def*)

apply(*simp add*:*Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def*

EvtSys-on-PIC-RGF-def Pre_{es}*-def getrgformula-def*)
apply *auto*[*1*]
apply(*case-tac k* = *Ctrl*)
apply (*simp add*:*EvtSys-on-Motor-RGF-def getrgformula-def*)
apply(*case-tac k* = *Radar*)
apply (*simp add*:*EvtSys-on-Radar-RGF-def getrgformula-def*)

apply(*case-tac k = PIC*)
apply (*simp add:EvtSys-on-PIC-RGF-def getrgformula-def*)
using *Device.exhaust* **apply** *blast*

apply *simp*

apply(*simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def*

EvtSys-on-PIC-RGF-def $Guar_{es}$*-def* $Rely_{es}$*-def getrgformula-def*)
apply *auto*[*1*]
apply(*case-tac j = Ctrl*)
apply(*case-tac k = Ctrl*)
apply *simp*
apply(*case-tac k = Radar*)
apply *auto*[*1*]
apply(*simp add: EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def*
getrgformula-def)
apply *auto*[*1*]
apply(*case-tac k = PIC*)
apply(*simp add: EvtSys-on-Motor-RGF-def EvtSys-on-PIC-RGF-def*
getrgformula-def)
using *Device.exhaust* **apply** *blast*
apply(*case-tac j = Radar*)
apply(*case-tac k = Radar*)
apply *simp*
apply(*case-tac k = Ctrl*)
apply *auto*[*1*]
apply(*simp add: EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def*
getrgformula-def)
apply *auto*[*1*]
apply(*case-tac k = PIC*)
apply(*simp add: EvtSys-on-PIC-RGF-def getrgformula-def*)
using *Device.exhaust* **apply** *blast*
apply(*case-tac j = PIC*)
apply(*case-tac k = PIC*)
apply *simp*
apply(*case-tac k = Ctrl*)
apply *auto*[*1*]
apply(*simp add: EvtSys-on-Motor-RGF-def EvtSys-on-PIC-RGF-def*
getrgformula-def)
apply (*metis* (*full-types*) *Irq.exhaust*)
apply(*case-tac k = Radar*)
apply *auto*[*1*]
apply(*simp add: EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def*
getrgformula-def)
apply(*case-tac a = b*)
apply *simp*
apply *simp*
apply (*erule exE*)

apply(*case-tac x = R*)
using *Irq.exhaust* **apply** *auto*[*1*]
using *Irq.exhaust* **apply** *auto*[*1*]
using *Device.exhaust* **apply** *blast*
using *Device.exhaust* **apply** *blast*

apply(*simp add*:*Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def*

EvtSys-on-PIC-RGF-def $Guar_{es}$*-def* $Rely_{es}$*-def getrgformula-def sys-guar-def*)
apply(*rule allI*)
apply(*case-tac k = PIC*) **apply** *clarsimp*
apply(*simp add*: *EvtSys-on-PIC-RGF-def getrgformula-def*) **apply** *blast*
apply(*case-tac k = Radar*)
apply(*simp add*: *EvtSys-on-Radar-RGF-def getrgformula-def*) **apply** *auto*[*1*]
apply(*case-tac k = Ctrl*)
apply(*simp add*: *EvtSys-on-Motor-RGF-def getrgformula-def*) **apply** *auto*[*1*]
apply (*meson Device.exhaust*)

by(*simp add*:*Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def*

EvtSys-on-PIC-RGF-def $Post_{es}$*-def getrgformula-def*)

5.4 Invariant proof

definition *invariant* :: (*State*) *invariant*
where *invariant s* ≡ ¬ *collide* (*car-pos s*) (*obstacle-pos s*)

lemma *init-sat-inv*: *invariant s0*
by(*simp add*:*s0-init init-def invariant-def collide-def*)

lemma *stb-guar-interrupt*: *stable* (*Collect invariant*) (⦃*hd* ᵒ*stack* ≠ *d* ∧ ᵃ*stack* = *d* # ᵒ*stack* ∧ ᵃ*car-pos* = ᵒ*car-pos*
∧ ᵃ*i* = ᵒ*i* ∧ ᵃ*pos-aux* = ᵒ*pos-aux* ∧ ᵃ*obstacle-pos* = ᵒ*obstacle-pos*
∧ ᵃ*obst-pos-aux* = ᵒ*obst-pos-aux*⦄ ∪ *Id*)
unfolding *stable-def invariant-def collide-def*
apply *clarify*
apply *simp*
by *auto*

lemma *stb-guar-forward*: *stable* (*Collect invariant*)
(⦃*hd* ᵒ*stack* = *M* ∧ (((ᵃ*i* = *0* ∨ ᵃ*i* = ᵒ*i* + *1* ∨ ᵃ*stack* = *tl* ᵒ*stack*) ∧ ᵃ*car-pos* = ᵒ*car-pos*) ∨
(¬*collide* (ᵒ*car-pos* + *1*) ᵒ*obstacle-pos* ∧ ᵃ*car-pos* = ᵒ*car-pos* + *1*))
∧ ᵃ*obstacle-pos* = ᵒ*obstacle-pos* ∧ ᵃ*obst-pos-aux* = ᵒ*obst-pos-aux*⦄ ∪ *Id*)
unfolding *stable-def invariant-def collide-def*
apply *clarify*
apply *simp*
by *auto*

lemma *stb-guar-backward*: *stable* (*Collect invariant*)
  ({|*hd* ^{o}*stack* = *M* ∧ (((^{a}*i* = *0* ∨ ^{a}*i* = ^{o}*i* + *1* ∨ ^{a}*stack* = *tl* ^{o}*stack*) ∧ ^{a}*car-pos* = ^{o}*car-pos*) ∨
    (¬*collide* (^{o}*car-pos* − *1*) ^{o}*obstacle-pos* ∧ ^{a}*car-pos* = ^{o}*car-pos* − *1*))
    ∧ ^{a}*obstacle-pos* = ^{o}*obstacle-pos* ∧ ^{a}*obst-pos-aux* = ^{o}*obst-pos-aux*|} ∪ *Id*)
 unfolding *stable-def invariant-def collide-def*
 apply *clarify*
 apply *simp*
 by *auto*

lemma *stb-guar-obstacle*: *stable* (*Collect invariant*)
 ({|*hd* ^{o}*stack* = *R* ∧ (((^{a}*stack* = *tl* ^{o}*stack* ∨ ^{a}*obst-pos-aux* = ^{o}*obstacle-pos*) ∧ ^{a}*obstacle-pos* = ^{o}*obstacle-pos*)
   ∨ (*set* ^{o}*obstacle-pos* ⊆ *set* ^{a}*obstacle-pos*
    ∧ *collide* (^{o}*car-pos* − *1*) ^{o}*obstacle-pos* = *collide* (^{a}*car-pos* − *1*) ^{a}*obstacle-pos*
     ∧ *collide* ^{o}*car-pos* ^{o}*obstacle-pos* = *collide* ^{a}*car-pos* ^{a}*obstacle-pos*
    ∧ *collide* (^{o}*car-pos* + *1*) ^{o}*obstacle-pos* = *collide* (^{a}*car-pos* + *1*) ^{a}*obstacle-pos*))
  ∧ ^{a}*car-pos* = ^{o}*car-pos* ∧ ^{a}*i* = ^{o}*i* ∧ ^{a}*pos-aux* = ^{o}*pos-aux* |} ∪ *Id*)
 unfolding *stable-def invariant-def collide-def*
 apply *clarify*
 apply *simp*
 by *auto*

theorem *invariant-presv-pares* Γ *invariant* (*paresys-spec Carsystem-Spec*) {*s0*} {}
apply(*rule invariant-theorem*[**where** *G*=*sys-guar* **and** *pst* = *UNIV*])
 using *functional-correctness* **apply** *fastforce*
 apply(*simp add*:*stable-def*)
 apply(*simp add*:*sys-guar-def*)
  apply(*rule stable-un-R*) **apply**(*rule stable-un-R*) **apply**(*rule stable-un-R*)
  using *stb-guar-forward* **apply** *fast*
  using *stb-guar-backward* **apply** *fast*
  using *stb-guar-obstacle* **apply** *fast*
  using *stb-guar-interrupt*
  apply (*metis* (*no-types*, *lifting*) *SUP-empty UN-simps*(*2*) *stable-id*
   stable-un-S sup-bot.right-neutral sup-commute)
  using *init-sat-inv* **apply** *simp*
done

end

6 Formal Specification and Reasoning of ARINC653 Multicore Microkernel

theory *ARINC653-MultiCore-QueIPC*
imports *../../Adapter-SIMP/picore-SIMP-lemma*

begin

6.1 functional specification

typedecl *Part*
typedecl *Sched*
typedecl *Message*
typedecl *Port*
typedecl *Core*

typedecl *QChannel*

```
record Config = c2s :: Core ⇒ Sched
                p2s :: Part ⇒ Sched
                p2p :: Port ⇒ Part
                chsrc :: QChannel ⇒ Port
                chdest :: QChannel ⇒ Port
                chmax :: QChannel ⇒ nat

axiomatization conf::Config
  where bij-c2s: bij (c2s conf)
    and portsrc-disj: ∀ c1 c2. c1 ≠ c2 ⟶ (chsrc conf) c1 ≠ (chsrc conf) c2
    and portdest-disj: ∀ c1 c2. c1 ≠ c2 ⟶ (chdest conf) c1 ≠ (chdest conf) c2
    and portsrcdest-disj: ∀ c1 c2. (chsrc conf) c1 ≠ (chdest conf) c2

lemma inj-surj-c2s: inj (c2s conf) ∧ surj (c2s conf)
  using bij-c2s by (simp add: bij-def)



definition is-src-qport :: Config ⇒ Port ⇒ bool
  where is-src-qport sc p ≡ (p∈range (chsrc sc))

definition is-dest-qport :: Config ⇒ Port ⇒ bool
  where is-dest-qport sc p ≡ (p∈range (chdest sc))

definition port-of-part :: Config ⇒ Port ⇒ Part ⇒ bool
  where port-of-part sc po pa ≡ ((p2p sc) po = pa)

definition ch-srcqport :: Config ⇒ Port ⇒ QChannel
  where ch-srcqport sc p ≡ SOME c. (chsrc sc) c = p

datatype PartMode = IDLE | READY | RUN

record State = cur :: Sched ⇒ Part option
               qbuf :: QChannel ⇒ Message list
               qbufsize :: QChannel ⇒ nat
               partst :: Part ⇒ PartMode
```

datatype *EL* = *Core-InitE* | *ScheduleE* | *Send-Que-MessageE* | *Recv-Que-MessageE*

datatype *parameter* = *Port Port* | *Message Message* | *Partition Part*

type-synonym *EventLabel* = *EL* × (*parameter list* × *Core*)

definition *get-evt-label* :: *EL* ⇒ *parameter list* ⇒ *Core* ⇒ *EventLabel* (- - @ - [*0*,*0*,*0*] *20*)
  where *get-evt-label el ps k* ≡ (*el*,(*ps*,*k*))

definition *Core-Init* :: *Core* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *event*

  where *Core-Init k*≡
    EVENT Core-InitE [] @ *k*
    THEN
      ´partst := (λ*p*. *if p2s conf p* = *c2s conf k* ∧ *´partst p* = *IDLE*
                  then READY else ´partst p)
    END

definition *System-Init* :: *Config* ⇒ (*State* × (*EventLabel*, *Core*, *State*, *State com option*) *x*)
  where *System-Init cfg* ≡ ((|*cur*=(λ*c*. *None*),
                          qbuf = (λ*c*. []),
                          qbufsize = (λ*c*. *0*),
                          partst = (λ*p*. *IDLE*)|),
                          (λ*k*. *Core-Init k*))

definition *Schedule* :: *Core* ⇒ *Part* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *event*
  where *Schedule k p* ≡
    EVENT ScheduleE [*Partition p*] @ *k*
    WHEN
      p2s conf p = *c2s conf k*
      ∧ (*´partst p* ≠ *IDLE*)
      ∧ (*´cur*((*c2s conf*) *k*) = *None*
          ∨ *p2s conf* (*the* (*´cur*((*c2s conf*) *k*))) = *c2s conf k*)
    THEN
      IF (*´cur*((*c2s conf*) *k*) ≠ *None*) *THEN*
        ATOMIC
          ´partst := *´partst*(*the* (*´cur* ((*c2s conf*) *k*)) := *READY*);;
          ´cur := *´cur*((*c2s conf*) *k* := *None*)
        END
      FI;;

      ATOMIC
        ´cur := *´cur*((*c2s conf*) *k* := *Some p*);;
        ´partst := *´partst*(*p* := *RUN*)
      END

END

definition *Send-Que-Message* :: *Core* ⇒ *Port* ⇒ *Message* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *event*
where *Send-Que-Message k p m* ≡
EVENT Send-Que-MessageE [*Port p*, *Message m*] @ *k*
WHEN
is-src-qport conf p
∧ ´*cur* ((*c2s conf*) *k*) ≠ *None*
∧ *port-of-part conf p* (*the* (´*cur* ((*c2s conf*) *k*)))
THEN
AWAIT ´*qbufsize* (*ch-srcqport conf p*) < *chmax conf* (*ch-srcqport conf p*) *THEN*
´*qbuf* := ´*qbuf* (*ch-srcqport conf p* := ´*qbuf* (*ch-srcqport conf p*) @ [*m*]);;
´*qbufsize* := ´*qbufsize* (*ch-srcqport conf p* := ´*qbufsize* (*ch-srcqport conf p*) + *1*)
END
END

definition *Recv-Que-Message* :: *Core* ⇒ *Port* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *event*
where *Recv-Que-Message k p* ≡
EVENT Recv-Que-MessageE [*Port p*] @ *k*
WHEN
is-dest-qport conf p
∧ ´*cur* ((*c2s conf*) *k*) ≠ *None*
∧ *port-of-part conf p* (*the* (´*cur* ((*c2s conf*) *k*)))
THEN
AWAIT ´*qbufsize* (*ch-srcqport conf p*) > *0 THEN*
´*qbuf* := ´*qbuf* (*ch-srcqport conf p* := *tl* (´*qbuf* (*ch-srcqport conf p*)));;
´*qbufsize* := ´*qbufsize* (*ch-srcqport conf p* := ´*qbufsize* (*ch-srcqport conf p*) − *1*)
END
END

6.2 Rely-guarantee condition of events

abbreviation *core-init-pre k* ≡ {|∀ *p*. *p2s conf p* = *c2s conf k* ⟶ ´*partst p* = *IDLE*|}
abbreviation *core-init-rely k* ≡ {|(∀ *p*. *p2s conf p* = *c2s conf k* ⟶ ª*partst p* = º*partst p*)|}
abbreviation *core-init-guar k* ≡ {|ª*cur*= º*cur* ∧ ª*qbuf*= º*qbuf* ∧ ª*qbufsize*= º*qbufsize*
∧ (∀ *p*. *p2s conf p* = *c2s conf k* ⟶ º*partst p* = *IDLE* ∧ ª*partst p* = *READY*)
∧ (∀ *c p*. *c* ≠ *k* ∧ *p2s conf p* = *c2s conf c* ⟶ ª*partst p* = º*partst p*)|} ∪ *Id*
abbreviation *core-init-post* ≡ {|*True*|}

definition *Core-Init-RGCond* :: *Core* $\Rightarrow$ (*State*) *PiCore-Hoare.rgformula*
where *Core-Init-RGCond k* $\equiv$ *RG*[*core-init-pre k*, *core-init-rely k*, *core-init-guar k*, *core-init-post*]

abbreviation *schedule-pre* $\equiv$ $\{\!|$ *True* $|\!\}$
abbreviation *schedule-rely k* $\equiv$
$\{\!|{}^{a}cur\ (c2s\ conf\ k) = {}^{o}cur\ \ (c2s\ conf\ k) \wedge$
$(\forall p.\ p2s\ conf\ p = c2s\ conf\ k \longrightarrow {}^{a}partst\ \ p = {}^{o}partst\ p)|\!\}$
abbreviation *schedule-guar k p* $\equiv$
$\{\!|({}^{a}cur = {}^{o}cur(c2s\ conf\ k := Some\ p)$
$\wedge\ {}^{a}partst = {}^{o}partst(the\ ({}^{a}cur(c2s\ conf\ k)) := RUN)$
$\wedge\ p2s\ conf\ p = c2s\ conf\ k$
$\vee\ ({}^{a}cur = {}^{o}cur(c2s\ conf\ k := None)$
$\wedge\ {}^{a}partst = {}^{o}partst(the\ ({}^{o}cur\ (c2s\ conf\ k)) := READY)))$
$\wedge\ (\forall c.\ c \neq k \longrightarrow {}^{a}cur\ (c2s\ conf\ c) = {}^{o}cur\ \ (c2s\ conf\ c))$
$\wedge\ (\forall c\ p.\ c \neq k \wedge p2s\ conf\ p = c2s\ conf\ c \longrightarrow {}^{a}partst\ \ p = {}^{o}partst\ p\ )$
$\wedge\ {}^{a}qbuf = {}^{o}qbuf$
$\wedge\ {}^{a}qbufsize = {}^{o}qbufsize|\!\} \cup Id$
abbreviation *schedule-post* $\equiv$ $\{\!|$ *True* $|\!\}$
definition *Schedule-RGCond* :: *Core* $\Rightarrow$ *Part* $\Rightarrow$ (*State*) *PiCore-Hoare.rgformula*
where *Schedule-RGCond k p* $\equiv$ *RG*[*schedule-pre*, *schedule-rely k*, *schedule-guar k p*, *schedule-post*]

abbreviation *snd-recv-pre* $\equiv$ $\{\!|$ *True* $|\!\}$
abbreviation *snd-recv-rely k* $\equiv$ $\{\!|{}^{a}cur\ (c2s\ conf\ k) = {}^{o}cur\ \ (c2s\ conf\ k)|\!\}$
abbreviation *snd-recv-guar p* $\equiv$
$\{\!|{}^{a}cur = {}^{o}cur \wedge {}^{a}partst = {}^{o}partst \wedge$
$({}^{o}qbufsize\ (ch\text{-}srcqport\ conf\ p) = length\ ({}^{o}qbuf\ \ (ch\text{-}srcqport\ conf\ p))$
$\longrightarrow {}^{a}qbufsize\ (ch\text{-}srcqport\ conf\ p) = length\ ({}^{a}qbuf\ \ (ch\text{-}srcqport\ conf\ p))) \wedge$
$(\forall c.\ c \neq ch\text{-}srcqport\ conf\ p \longrightarrow {}^{a}qbuf\ c = {}^{o}qbuf\ c) \wedge$
$(\forall c.\ c \neq ch\text{-}srcqport\ conf\ p \longrightarrow {}^{a}qbufsize\ c = {}^{o}qbufsize\ c)|\!\}$
abbreviation *snd-recv-post* $\equiv$ $\{\!|$ *True* $|\!\}$

definition *Send-Que-Message-RGCond* :: *Core* $\Rightarrow$ *Port* $\Rightarrow$ *Message* $\Rightarrow$ (*State*) *PiCore-Hoare.rgformula*
where *Send-Que-Message-RGCond k p m* $\equiv$ *RG*[*snd-recv-pre*, *snd-recv-rely k*, *snd-recv-guar p*, *snd-recv-post*]

definition *Recv-Que-Message-RGCond* :: *Core* $\Rightarrow$ *Port* $\Rightarrow$ (*State*) *PiCore-Hoare.rgformula*
where *Recv-Que-Message-RGCond k p* $\equiv$ *RG*[*snd-recv-pre*, *snd-recv-rely k*, *snd-recv-guar p*, *snd-recv-post*]

definition *Core-Init-RGF* :: *Core* $\Rightarrow$ (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-e*
where *Core-Init-RGF k* $\equiv$ (*Core-Init k*, *Core-Init-RGCond k*)

definition *Schedule-RGF* :: *Core* $\Rightarrow$ *Part* $\Rightarrow$ (*EventLabel*, *Core*, *State*, *State com*

option) *rgformula-e*
where *Schedule-RGF k p* ≡ (*Schedule k p*, *Schedule-RGCond k p*)

definition *Send-Que-Message-RGF* :: *Core* ⇒ *Port* ⇒ *Message* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-e*
where *Send-Que-Message-RGF k p m* ≡ (*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)

definition *Recv-Que-Message-RGF* :: *Core* ⇒ *Port* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-e*
where *Recv-Que-Message-RGF k p* ≡ (*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)

definition *EvtSys1-on-Core-RGF* :: *Core* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-es*
where *EvtSys1-on-Core-RGF k* ≡
(*rgf-EvtSys* (⋃ *p*.{*Schedule-RGF k p*} ∪
(⋃ (*p*, *m*). {*Send-Que-Message-RGF k p m*}) ∪
(⋃ *p*.{*Recv-Que-Message-RGF k p*})),
RG[⦃*True*⦄,
schedule-rely k,
((⋃ *p*. *schedule-guar k p*) ∪ (⋃ *p*. *snd-recv-guar p*) ∪ *Id*),
⦃*True*⦄])

definition *EvtSys-on-Core-RGF* :: *Core* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-es*
where *EvtSys-on-Core-RGF k* ≡
(*rgf-EvtSeq* (*Core-Init-RGF k*) (*EvtSys1-on-Core-RGF k*),
RG[*core-init-pre k*,
schedule-rely k,
((⋃ *p*. *schedule-guar k p*) ∪ (⋃ *p*. *snd-recv-guar p*) ∪ *Id* ∪ (*core-init-guar k*)),
⦃*True*⦄])

definition *ARINCXKernel-Spec* :: (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-par*
where *ARINCXKernel-Spec* ≡ (λ*k*. *EvtSys-on-Core-RGF k*)

consts *s0*::*State*
definition *s0-witness*::*State*
where *s0-witness* ≡ *fst* (*System-Init conf*)

specification (*s0*)
s0-init: *s0* ≡ *fst* (*System-Init conf*)
by *simp*

6.3 Functional correctness by rely guarantee proof

lemma *Core-Init-SatRG*: $\forall k.\ \Gamma\ (Core\text{-}Init\ k) \vdash Core\text{-}Init\text{-}RGCond\ k$
  apply(*simp add*:*Evt-sat-RG-def*)
  apply(*rule allI*)
  apply(*simp add*:*Core-Init-def*)
  apply(*rule BasicEvt*)
    apply(*simp add*:*body-def Core-Init-RGCond-def* Pre_f*-def* $Post_f$*-def*
        $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
    apply(*rule Basic*)
    unfolding *guard-def* **apply** *simp*
    apply *simp*
    apply *auto*
    using *inj-surj-c2s injI surj-def* **apply** (*simp add*: *inj-eq*)
    apply(*simp add*:*stable-def*)+
    apply(*simp add*:*Core-Init-RGCond-def* Pre_f*-def* $Post_f$*-def* $Guar_f$*-def*
        $Rely_f$*-def getrgformula-def guard-def stable-def*)
    apply(*simp add*:*Core-Init-RGCond-def* $Guar_f$*-def getrgformula-def stable-def*)
  done

lemma *Sched-SatRG-h2*:
  $\vdash_I$ *Some* (´*cur* := ´*cur*(*c2s conf k* $\mapsto$ *p*);;
    ´*partst* := ´*partst* (*p* := *RUN*))
   sat_p [⦃*p2s conf p* = *c2s conf k* $\wedge$ ´*cur* (*c2s conf k*) = *None*⦄ $\cap$ {*V*},
     {(*s*, *t*). *s* = *t*}, *UNIV*,
     ⦃(´*cur* = *cur V*(*c2s conf k* $\mapsto$ *p*) $\wedge$
       ´*partst* = (*partst V*)(*the* (´*cur* (*c2s conf k*)) := *RUN*) $\wedge$
       p2s conf p = *c2s conf k* $\vee$
       ´*cur* = (*cur V*)(*c2s conf k* := *None*) $\wedge$
       ´*partst* = (*partst V*)(*the* (*cur V* (*c2s conf k*)) := *READY*)) $\wedge$
      ($\forall$ *c*. *c* $\neq$ *k* $\longrightarrow$ ´*cur* (*c2s conf c*) = *cur V* (*c2s conf c*)) $\wedge$
      ($\forall$ *c p*. *c* $\neq$ *k* $\wedge$ *p2s conf p* = *c2s conf c* $\longrightarrow$ ´*partst p* = *partst V p*) $\wedge$
      ´*qbuf* = *qbuf V* $\wedge$ ´*qbufsize* = *qbufsize V* $\vee$
      ´((=) *V*)⦄]
  apply(*case-tac p2s conf p* = *c2s conf k* $\wedge$ (*cur V*) (*c2s conf k*) = *None*)
    apply *simp*
    apply(*rule Seq*[**where** *mid*={*s*. *s* = *V* ⦇ *cur* := (*cur V*) (*c2s conf k* := *Some p*)⦈}])
      apply(*rule Basic*)
        apply *auto*[*1*]
        apply(*simp add*:*stable-def*)+
      apply(*rule Basic*)
        apply *simp*
        apply(*rule disjI1*)
        using *inj-surj-c2s injI surj-def* **apply** (*simp add*: *inj-eq*)
        apply(*simp add*:*stable-def*)+ **apply** *auto*[*1*]
    apply(*rule Seq*[**where** *mid*={}])
      apply(*rule Basic*)
        apply(*simp add*:*stable-def*)+

```
      apply(rule Basic)
        apply(simp add:stable-def)+ apply auto[1]
  done

lemma Sched-SatRG-h1:
   ⊢_I Some (´partst := ´partst(the (´cur (c2s conf k)) := READY);;
      ´cur := ´cur (c2s conf k := None))
      sat_p [⦃p2s conf p = c2s conf k ∧ ´partst p ≠ IDLE ∧ (´cur (c2s conf k) =
None
            ∨ p2s conf (the (´cur (c2s conf k))) = c2s conf k)⦄ ∩
                ⦃∃ y. ´cur (c2s conf k) = Some y⦄ ∩ {V},
          {(s, t). s = t}, UNIV,
          ⦃(´cur = cur V(c2s conf k ↦ p) ∧
              ´partst = (partst V)(the (´cur (c2s conf k)) := RUN) ∧
              p2s conf p = c2s conf k ∨
              ´cur = (cur V)(c2s conf k := None) ∧
              ´partst = (partst V)(the (cur V (c2s conf k)) := READY)) ∧
            (∀ c. c ≠ k ⟶ ´cur (c2s conf c) = cur V (c2s conf c)) ∧
            (∀ c p. c ≠ k ∧ p2s conf p = c2s conf c ⟶ ´partst p = partst V p) ∧
            ´qbuf = qbuf V ∧ ´qbufsize = qbufsize V ∨
            ´((=) V)⦄ ∩
           ⦃p2s conf p = c2s conf k ∧ ´cur (c2s conf k) = None⦄]
   apply(case-tac p2s conf p = c2s conf k ∧ partst V p ≠ IDLE
         ∧ ((cur V) (c2s conf k) = None ∨ p2s conf (the ((cur V) (c2s conf k)))
= c2s conf k)
           ∧ (∃ y. (cur V) (c2s conf k) = Some y))
    apply simp
    apply(rule Seq[where mid={s. s = V (| partst := (partst V) (the ((cur V)
(c2s conf k)) := READY)|)
                                   ∧ p2s conf p = c2s conf k}])
      apply simp
      apply(rule Basic)
        apply auto[1]
        apply(simp add:stable-def)+
      apply(rule Basic)
        apply simp
        apply(rule disjI1)
          apply(rule conjI)
            using inj-surj-c2s injI surj-def apply (simp add: inj-eq)
            apply(rule impI)
            apply(case-tac cur V (c2s conf k) = None)
              apply simp
              using inj-surj-c2s injI surj-def apply (simp add: inj-eq)
        apply simp
        apply(simp add:stable-def)
        apply(simp add:stable-def) apply auto[1]
    apply(rule Seq[where mid={}])
      apply(rule Basic)
        apply(simp add:stable-def)+
```

apply(*rule Basic*)
apply(*simp add*:*stable-def*)+
apply *auto*[*1*]
done

lemma *Sched-SatRG*: Γ (*Schedule k p*) $\vdash$ *Schedule-RGCond k p*
apply(*simp add*:*Evt-sat-RG-def*)
apply(*simp add*:*Schedule-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*body-def Schedule-RGCond-def guard-def* Pre_f*-def* $Post_f$*-def* $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*rule Seq*[**where** *mid*=⦃*p2s conf p* = *c2s conf k* $\wedge$ ´*cur*(*c2s conf k*) = *None* ⦄])
apply(*rule Cond*)
apply(*simp add*: *stable-def*)
apply(*rule Await*)
apply(*simp add*: *stable-def*)+
apply(*rule allI*) **apply**(*rule Sched-SatRG-h1*)
apply(*simp add*: *Skip-def*)
apply(*rule Basic*)
apply *auto*[*1*]
apply *auto*[*1*]
apply(*simp add*: *stable-def*)+
apply(*rule Await*)
apply(*simp add*: *stable-def*)+
apply(*rule allI*) **apply**(*rule Sched-SatRG-h2*)
apply(*simp add*: *stable-def Schedule-RGCond-def* Pre_f*-def* $Post_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*simp add*: *Schedule-RGCond-def* Pre_f*-def* $Post_f$*-def* $Guar_f$*-def getrgformula-def*)
done

lemma *Send-Que-Message-SatRG-h1*:
$\vdash_I$ *Some* (´*qbuf* := ´*qbuf*(*ch-srcqport conf p* := ´*qbuf* (*ch-srcqport conf p*) @ [*m*]);;
´*qbufsize* := ´*qbufsize* (*ch-srcqport conf p* :=
Suc (´*qbufsize* (*ch-srcqport conf p*))))
sat_p [⦃*is-src-qport conf p* $\wedge$ ($\exists$ *y*. ´*cur* ((*c2s conf*) *k*) = *Some y*)
$\wedge$ *port-of-part conf p* (*the* (´*cur* ((*c2s conf*) *k*)))⦄ $\cap$
⦃´*qbufsize* (*ch-srcqport conf p*) < *chmax conf* (*ch-srcqport conf p*)⦄ $\cap$ {*V*},
{(*s*, *t*). *s* = *t*}, *UNIV*,
⦃´(*Pair V*) $\in$ ⦃^{a}*cur* = ^{o}*cur* $\wedge$
^{a}*partst* = ^{o}*partst* $\wedge$
(^{o}*qbufsize* (*ch-srcqport conf p*) =
length (^{o}*qbuf* (*ch-srcqport conf p*)) $\longrightarrow$
^{a}*qbufsize* (*ch-srcqport conf p*) =
length (^{a}*qbuf* (*ch-srcqport conf p*))) $\wedge$
($\forall$ *c*. *c* $\neq$ *ch-srcqport conf p* $\longrightarrow$ ^{a}*qbuf c* = ^{o}*qbuf c*) $\wedge$
($\forall$ *c*. *c* $\neq$ *ch-srcqport conf p* $\longrightarrow$ ^{a}*qbufsize c* = ^{o}*qbufsize c*)⦄⦄ $\cap$

```
UNIV]
  apply(case-tac is-src-qport conf p ∧ (∃ y. (cur V) ((c2s conf) k) = Some y)
                  ∧ port-of-part conf p (the ((cur V) ((c2s conf) k)))
                  ∧ (qbufsize V) (ch-srcqport conf p) < chmax conf (ch-srcqport conf
p))
    apply simp
    apply(rule Seq[where mid={s. s = V(|qbuf := (qbuf V)(ch-srcqport conf p
                                := (qbuf V) (ch-srcqport conf p) @ [m])|)}])
      apply(rule Basic)
        apply auto[1]
        apply(simp add: stable-def)+
      apply(rule Basic)
        apply auto[1]
        apply(simp add: stable-def)+
    apply(rule Seq[where mid={}])
      apply(rule Basic)
        apply(simp add:stable-def)+
      apply(rule Basic)
        apply(simp add:stable-def)+
  done


lemma Send-Que-Message-SatRG:
  Γ (Send-Que-Message k p m) ⊢ Send-Que-Message-RGCond k p m
  apply(simp add:Evt-sat-RG-def)
  apply(simp add:Send-Que-Message-def)
  apply(rule BasicEvt)
  apply(simp add:body-def Send-Que-Message-RGCond-def guard-def Pre_f-def
          Post_f-def Rely_f-def Guar_f-def getrgformula-def)
    apply(rule Await)
      apply(simp add: stable-def)
      apply(simp add: stable-def)
      apply(rule allI) apply(rule Send-Que-Message-SatRG-h1)
   apply(simp add: stable-def Send-Que-Message-RGCond-def Pre_f-def Rely_f-def
getrgformula-def)
  apply(simp add: Send-Que-Message-RGCond-def Guar_f-def getrgformula-def)
  done


lemma Recv-Que-Message-SatRG-h1:
   ⊢_I Some (´qbuf := ´qbuf(ch-srcqport conf p := tl (´qbuf (ch-srcqport conf p)));;
       ´qbufsize := ´qbufsize (ch-srcqport conf p := ´qbufsize (ch-srcqport conf p) −
Suc 0))
     sat_p [{|is-dest-qport conf p ∧ (∃ y. ´cur ((c2s conf) k) = Some y)
             ∧ port-of-part conf p (the (´cur ((c2s conf) k)))|} ∩
            {|0 < ´qbufsize (ch-srcqport conf p)|} ∩ {V},
          {(s, t). s = t}, UNIV,
          {|´(Pair V) ∈ {|ªcur = ºcur ∧ ªpartst = ºpartst ∧
                          (ºqbufsize (ch-srcqport conf p) = length (ºqbuf (ch-srcqport
```

```
conf p)) ⟶
                              ᵃqbufsize (ch-srcqport conf p) = length (ᵃqbuf (ch-srcqport
conf p))) ∧
                         (∀ c. c ≠ ch-srcqport conf p ⟶ ᵃqbuf c = ᵒqbuf c) ∧
                        (∀ c. c ≠ ch-srcqport conf p ⟶ ᵃqbufsize c = ᵒqbufsize c)⦄⦄ ∩
UNIV]
  apply(case-tac is-dest-qport conf p ∧ (∃ y. (cur V) ((c2s conf) k) = Some y)
                ∧ port-of-part conf p (the ((cur V) ((c2s conf) k)))
                ∧ 0 < (qbufsize V) (ch-srcqport conf p))
    apply simp
    apply(rule Seq[where mid={s. s = V⦇qbuf := (qbuf V)(ch-srcqport conf p :=
tl ((qbuf V) (ch-srcqport conf p)))⦈}])
      apply(rule Basic)
        apply auto[1]
        apply(simp add: stable-def)+
      apply(rule Basic)
        apply auto[1]
        apply(simp add: stable-def)+
    apply(rule Seq[where mid={}])
      apply(rule Basic)
        apply(simp add:stable-def)+
      apply(rule Basic)
        apply(simp add:stable-def)+
  done


lemma Recv-Que-Message-SatRG: Γ (Recv-Que-Message k p) ⊢ Recv-Que-Message-RGCond
k p
  apply(simp add:Evt-sat-RG-def)
  apply(simp add:Recv-Que-Message-def)
  apply(rule BasicEvt)
  apply(simp add:body-def Recv-Que-Message-RGCond-def guard-def Pre_f-def
           Post_f-def Rely_f-def Guar_f-def getrgformula-def)
    apply(rule Await)
      apply(simp add: stable-def)+
      apply(rule allI) using Recv-Que-Message-SatRG-h1 apply fastforce
   apply(simp add: stable-def Recv-Que-Message-RGCond-def Pre_f-def Rely_f-def
getrgformula-def)
  apply(simp add: Recv-Que-Message-RGCond-def Guar_f-def getrgformula-def)
  done


lemma EvtSys1-on-core-SatRG:
  ∀ k. Γ ⊢ fst (EvtSys1-on-Core-RGF k) sat_s
             [Pre_f (snd (EvtSys1-on-Core-RGF k)),
              Rely_f (snd (EvtSys1-on-Core-RGF k)),
              Guar_f (snd (EvtSys1-on-Core-RGF k)),
              Post_f (snd (EvtSys1-on-Core-RGF k))]
  apply(rule allI)
```

apply(*simp add*:*EvtSys1-on-Core-RGF-def* Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def getrgformula-def*)
apply(*rule EvtSys-h*)

apply(*clarify*)
apply(*case-tac* $(a,b) \in \{(Schedule\text{-}RGF\ k\ x)\}$)
using *Sched-SatRG Schedule-RGF-def Evt-sat-RG-def* E_e*-def* Pre_e*-def* $Rely_e$*-def* $Guar_e$*-def* $Post_e$*-def*
$Guar_f$*-def* $Post_f$*-def* Pre_f*-def* $Rely_f$*-def snd-conv fst-conv* **apply** (*metis singletonD*)
apply(*case-tac* $(a,b) \in (\bigcup (p,\ m).\ \{Send\text{-}Que\text{-}Message\text{-}RGF\ k\ p\ m\})$)
apply(*clarify*)
using *Send-Que-Message-SatRG Send-Que-Message-RGF-def* E_e*-def* Pre_e*-def* $Rely_e$*-def* $Guar_e$*-def* $Post_e$*-def*
$Guar_f$*-def* $Post_f$*-def* Pre_f*-def* $Rely_f$*-def snd-conv fst-conv Evt-sat-RG-def* **apply** *metis*
apply(*case-tac* $(a,b) \in (\bigcup p.\ \{Recv\text{-}Que\text{-}Message\text{-}RGF\ k\ p\})$)
apply(*clarify*)
using *Recv-Que-Message-SatRG Recv-Que-Message-RGF-def* E_e*-def* Pre_e*-def* $Rely_e$*-def* $Guar_e$*-def* $Post_e$*-def*
$Guar_f$*-def* $Post_f$*-def* Pre_f*-def* $Rely_f$*-def snd-conv fst-conv Evt-sat-RG-def* **apply** *metis*
apply *blast*

apply(*clarify*)
apply(*case-tac* $(a,b) \in \{(Schedule\text{-}RGF\ k\ x)\}$)
apply(*simp add*:*Schedule-RGF-def Schedule-RGCond-def* Pre_e*-def getrgformula-def*)
apply(*case-tac* $(a,b) \in (\bigcup (p,\ m).\ \{Send\text{-}Que\text{-}Message\text{-}RGF\ k\ p\ m\})$)
apply *clarify*
apply(*simp add*:*Send-Que-Message-RGF-def Send-Que-Message-RGCond-def* Pre_e*-def getrgformula-def*)
apply(*case-tac* $(a,b) \in (\bigcup p.\ \{Recv\text{-}Que\text{-}Message\text{-}RGF\ k\ p\})$)
apply(*clarify*)
apply(*simp add*:*Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def* Pre_e*-def getrgformula-def*)
apply *blast*

apply(*clarify*)
apply(*case-tac* $(a,b) \in \{(Schedule\text{-}RGF\ k\ x)\}$)
apply(*simp add*:*Schedule-RGF-def Schedule-RGCond-def* $Rely_e$*-def getrgformula-def*)

apply(*case-tac* $(a,b) \in (\bigcup (p,\ m).\ \{Send\text{-}Que\text{-}Message\text{-}RGF\ k\ p\ m\})$)
apply *clarify*
apply(*simp add*:*Send-Que-Message-RGF-def Send-Que-Message-RGCond-def* $Rely_e$*-def getrgformula-def*)
apply(*case-tac* $(a,b) \in (\bigcup p.\ \{Recv\text{-}Que\text{-}Message\text{-}RGF\ k\ p\})$)
apply(*clarify*)
apply(*simp add*:*Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def* $Rely_e$*-def getrgformula-def*)

apply *blast*

apply(*clarify*)
apply(*case-tac* $(a,b) \in \{(Schedule\text{-}RGF\ k\ x)\}$)
apply(*simp add*:*Schedule-RGF-def Schedule-RGCond-def getrgformula-def* $Guar_e$*-def*)

apply *auto*[*1*]
apply(*case-tac* $(a,b) \in (\bigcup (p,\ m).\ \{Send\text{-}Que\text{-}Message\text{-}RGF\ k\ p\ m\})$)
apply(*simp add*:*Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def* $Guar_e$*-def*)
apply *auto*[*1*]
apply(*case-tac* $(a,b) \in (\bigcup p.\ \{Recv\text{-}Que\text{-}Message\text{-}RGF\ k\ p\})$)
apply(*simp add*:*Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def* $Guar_e$*-def*)
apply *auto*[*1*]
apply *blast*

apply(*clarify*)
apply(*case-tac* $(a,b) \in \{(Schedule\text{-}RGF\ k\ x)\}$)
apply(*simp add*:*Schedule-RGF-def Schedule-RGCond-def getrgformula-def* $Guar_e$*-def*)
apply(*case-tac* $(a,b) \in (\bigcup (p,\ m).\ \{Send\text{-}Que\text{-}Message\text{-}RGF\ k\ p\ m\})$)
apply(*simp add*:*Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def* $Guar_e$*-def*)
apply(*case-tac* $(a,b) \in (\bigcup p.\ \{Recv\text{-}Que\text{-}Message\text{-}RGF\ k\ p\})$)
apply(*simp add*:*Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def* $Guar_e$*-def*)
apply *blast*

apply(*clarify*)
apply(*case-tac* $(a,b) \in \{(Schedule\text{-}RGF\ k\ xa)\}$)
apply(*case-tac* $(aa,ba) \in \{(Schedule\text{-}RGF\ k\ xb)\}$)
apply(*simp add*:*Schedule-RGF-def Schedule-RGCond-def getrgformula-def* Pre_e*-def*)
apply(*case-tac* $(aa,ba) \in (\bigcup (p,\ m).\ \{Send\text{-}Que\text{-}Message\text{-}RGF\ k\ p\ m\})$)
apply(*simp add*:*Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def* Pre_e*-def*)
apply *auto*[*1*]
apply(*case-tac* $(aa,ba) \in (\bigcup p.\ \{Recv\text{-}Que\text{-}Message\text{-}RGF\ k\ p\})$)
apply(*simp add*:*Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def* Pre_e*-def*)
apply *auto*[*1*]
apply *blast*

apply(*case-tac* $(a,b) \in (\bigcup (p,\ m).\ \{Send\text{-}Que\text{-}Message\text{-}RGF\ k\ p\ m\})$)
apply(*case-tac* $(aa,ba) \in \{(Schedule\text{-}RGF\ k\ xb)\}$)
apply(*simp add*:*Schedule-RGF-def Schedule-RGCond-def getrgformula-def* Pre_e*-def*)
apply(*case-tac* $(aa,ba) \in (\bigcup (p,\ m).\ \{Send\text{-}Que\text{-}Message\text{-}RGF\ k\ p\ m\})$)
apply(*simp add*:*Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def* Pre_e*-def*)
apply *auto*[*1*]

apply(*case-tac* $(aa,ba) \in (\bigcup p.\ \{$*Recv-Que-Message-RGF k p*$\})$)
apply(*simp add:Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def* Pre_e*-def*)
apply *auto*[*1*]
apply *blast*
apply(*case-tac* $(a,b) \in (\bigcup p.\ \{$*Recv-Que-Message-RGF k p*$\})$)
apply(*case-tac* $(aa,ba) \in \{($*Schedule-RGF k xb*$)\}$)
apply(*simp add:Schedule-RGF-def Schedule-RGCond-def getrgformula-def* Pre_e*-def*)
apply(*case-tac* $(aa,ba) \in (\bigcup (p,\ m).\ \{$*Send-Que-Message-RGF k p m*$\})$)
apply(*simp add:Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def* Pre_e*-def*)
apply *auto*[*1*]
apply(*case-tac* $(aa,ba) \in (\bigcup p.\ \{$*Recv-Que-Message-RGF k p*$\})$)
apply(*simp add:Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def* Pre_e*-def*)
apply *auto*[*1*]
apply *blast*
apply *blast*
apply (*simp add:stable-def*)
by *simp*

lemma *EvtSys-on-core-SatRG*:
$\forall k.\ \Gamma \vdash$ *fst* (*EvtSys-on-Core-RGF k*) sat_s
[Pre_f (*snd* (*EvtSys-on-Core-RGF k*)),
$Rely_f$ (*snd* (*EvtSys-on-Core-RGF k*)),
$Guar_f$ (*snd* (*EvtSys-on-Core-RGF k*)),
$Post_f$ (*snd* (*EvtSys-on-Core-RGF k*))]
apply(*rule allI*)
apply(*simp add:EvtSys-on-Core-RGF-def* Pre_f*-def* $Rely_f$*-def*
$Guar_f$*-def* $Post_f$*-def getrgformula-def*)
apply(*rule EvtSeq-h*)

apply(*simp add:*E_e*-def Core-Init-RGF-def* Pre_e*-def* $Rely_e$*-def* $Guar_e$*-def* $Post_e$*-def*)
using *Core-Init-SatRG getrgformula-def*
apply (*simp add: Evt-sat-RG-def* $Guar_f$*-def* $Post_f$*-def* Pre_f*-def* $Rely_f$*-def*)
apply *fastforce*
using *EvtSys1-on-core-SatRG* **apply** *fastforce*
apply(*simp add:Core-Init-RGF-def Core-Init-RGCond-def* Pre_e*-def getrgformula-def*)

apply(*simp add:EvtSys1-on-Core-RGF-def* $Post_f$*-def getrgformula-def*)
apply(*simp add:Core-Init-RGF-def Core-Init-RGCond-def* $Rely_e$*-def getrgformula-def*)

apply *auto*[*1*]
apply(*simp add:EvtSys1-on-Core-RGF-def* $Rely_f$*-def getrgformula-def Core-Init-RGF-def*)

apply(*simp add:Core-Init-RGF-def Core-Init-RGCond-def* $Guar_e$*-def* $Guar_f$*-def*

getrgformula-def EvtSys1-on-Core-RGF-def)
apply(*simp add:EvtSys1-on-Core-RGF-def Core-Init-RGCond-def* $Guar_f$*-def getrgformula-def*)

by (*simp add*:*EvtSys1-on-Core-RGF-def Core-Init-RGF-def Core-Init-RGCond-def*

$Post_e$*-def* Pre_f*-def getrgformula-def*)

definition *sys-guar* $\equiv \bigcup k.$ (($\bigcup p.$ *schedule-guar k p*) $\cup$ ($\bigcup p.$ *snd-recv-guar p*) $\cup$ *Id* $\cup$ (*core-init-guar k*))

lemma *esys-guar-in-sys*: $Guar_{es}$ (*ARINCXKernel-Spec k*) $\subseteq$ *sys-guar*
apply(*simp add*:$Guar_{es}$*-def ARINCXKernel-Spec-def EvtSys-on-Core-RGF-def getrgformula-def sys-guar-def*)
by *auto*

lemma *spec-sat-rg*: $\Gamma \vdash$ *ARINCXKernel-Spec SAT* [{*s0*}, {}, *sys-guar*, *UNIV*]
apply (*rule ParallelESys*)
apply(*simp add*:*ARINCXKernel-Spec-def*) **using** *EvtSys-on-core-SatRG*
apply (*simp add*: $Guar_{es}$*-def* $Guar_f$*-def* $Post_{es}$*-def* $Post_f$*-def* Pre_{es}*-def* Pre_f*-def* $Rely_{es}$*-def* $Rely_f$*-def*)
apply *fastforce*
apply(*simp add*:*ARINCXKernel-Spec-def EvtSys-on-Core-RGF-def* Pre_{es}*-def getrgformula-def s0-def System-Init-def*)
apply *simp*

apply *clarsimp*
apply(*simp add*:*ARINCXKernel-Spec-def EvtSys-on-Core-RGF-def*
$Guar_{es}$*-def* $Rely_{es}$*-def getrgformula-def*)
apply *auto*[*1*] **apply** *metis* **apply** *metis* **apply** *metis*
using *esys-guar-in-sys* **apply** *fast*
apply *simp+*
done

6.4 Invariant proof

definition *cur-part-cond* :: *State* $\Rightarrow$ *bool*
where *cur-part-cond s* $\equiv \forall$ *sched p*. (*cur s*) *sched* = *Some p* $\longrightarrow$ *sched* = (*p2s conf*) *p*

definition *cur-part-mode-cond* :: *State* $\Rightarrow$ *bool*
where *cur-part-mode-cond s* $\equiv$
$\forall$ *sched p*. *p2s conf p* = *sched* $\wedge$ (*cur s*) *sched* = *Some p* $\longrightarrow$ (*partst s*) *p* = *RUN*

definition *qbuf-size-cond* :: *State* $\Rightarrow$ *bool*
where *qbuf-size-cond s* $\equiv \forall$ *c*. (*qbufsize s*) *c* = *length* ((*qbuf s*) *c*)

definition *invariant s* $\equiv$ *cur-part-cond s* $\wedge$ *cur-part-mode-cond s* $\wedge$ *qbuf-size-cond s*

lemma *init-sat-inv*: *invariant s0*

```
  by(simp add:s0-init System-Init-def invariant-def cur-part-cond-def
        cur-part-mode-cond-def qbuf-size-cond-def)


lemma stb-guar-coreinit: stable (Collect invariant) (core-init-guar k)
  unfolding stable-def invariant-def cur-part-cond-def
        cur-part-mode-cond-def qbuf-size-cond-def
  apply clarify
  apply simp
  apply (rule conjI)
    apply(rule allI)+ apply(rule impI) apply presburger
    apply(rule conjI)
      apply(rule allI)+ apply(rule impI)
      apply(case-tac x = y)
        apply blast
        apply(case-tac p2s conf p = c2s conf k)
          apply (metis PartMode.distinct(3))
          apply (metis (no-types, lifting) inj-surj-c2s surj-def)
      apply(rule allI) by metis

lemma stb-guar-sched: stable (Collect invariant) (schedule-guar k p)
  apply(simp add:stable-def invariant-def cur-part-cond-def
        cur-part-mode-cond-def qbuf-size-cond-def)
  apply(rule allI)
  apply(rule impI)
  apply(rule allI)
  apply(rule conjI)
    apply(rule impI)
    apply(rule conjI)
      apply(rule allI)+
      apply(rule impI)
        apply auto[1]
        apply (metis option.sel)
        apply (metis option.discI)
    apply(rule allI)+
    apply(rule impI)
    apply(case-tac p2s conf pa = c2s conf k)
      apply auto[1]
      apply (metis (no-types, lifting) inj-surj-c2s surj-def)
    apply(rule impI)
    apply(rule conjI)
      apply blast
      apply(rule allI)
      by simp

lemma stb-guar-sndrecvmsg:
  stable (Collect invariant) (snd-recv-guar p)
  apply(simp add:stable-def invariant-def cur-part-cond-def qbuf-size-cond-def)
  apply(simp add:cur-part-mode-cond-def)
```

apply(*rule allI*) **apply**(*rule impI*)
apply(*rule allI*) **apply**(*rule impI*)
apply(*rule allI*) **by** *metis*

lemma *stb-pred-rel*: *stable* (*Collect P*) *RG* $\Longrightarrow$ (*s*, *r*) $\in$ *RG* $\Longrightarrow$ *P s* $\Longrightarrow$ *P r*
by(*simp add:stable-def*)

lemma *core-init-g-stb-inv*: (*s*,*r*)$\in$*core-init-guar k* $\Longrightarrow$ *invariant s* $\Longrightarrow$ *invariant r*
using *stb-guar-coreinit*[*of k*] *stb-pred-rel*[*of invariant core-init-guar k s r*] **by** *fast*

lemma *sched-g-stb-inv*: (*s*,*r*)$\in$*schedule-guar k p* $\Longrightarrow$ *invariant s* $\Longrightarrow$ *invariant r*
using *stb-guar-sched*[*of k*] *stb-pred-rel*[*of invariant schedule-guar k p s r*] **by** *fast*

lemma *snd-recv-g-stb-inv*: (*s*,*r*)$\in$*snd-recv-guar p* $\Longrightarrow$ *invariant s* $\Longrightarrow$ *invariant r*
using *stb-guar-sndrecvmsg*[*of p*] *stb-pred-rel*[*of invariant snd-recv-guar p s r*] **by** *fast*

theorem *invariant-presv-pares* Γ *invariant* (*paresys-spec ARINCXKernel-Spec*) {*s0*} {}
apply(*rule invariant-theorem*[**where** *G*=*sys-guar* **and** *pst* = *UNIV*])
using *spec-sat-rg* **apply** *fast*
apply(*simp add:stable-def*)
apply(*simp add:sys-guar-def*)
apply(*rule stable-un-S*) **apply** *clarify*
apply(*rule stable-un-R*) **apply**(*rule stable-un-R*) **apply**(*rule stable-un-R*)
using *stb-guar-sched* **apply** (*smt stable-def UN-iff Un-iff mem-Collect-eq*)
apply(*rule stable-un-S*) **apply** *clarify* **using** *stb-guar-sndrecvmsg* **apply** *fast*
apply(*simp add:stable-def*)
using *stb-guar-coreinit* **apply** *fast*
using *init-sat-inv* **apply** *simp*
done

end