

Hoare Logic for Parallel Programs

Leonor Prensa Nieto

June 8, 2008

Abstract

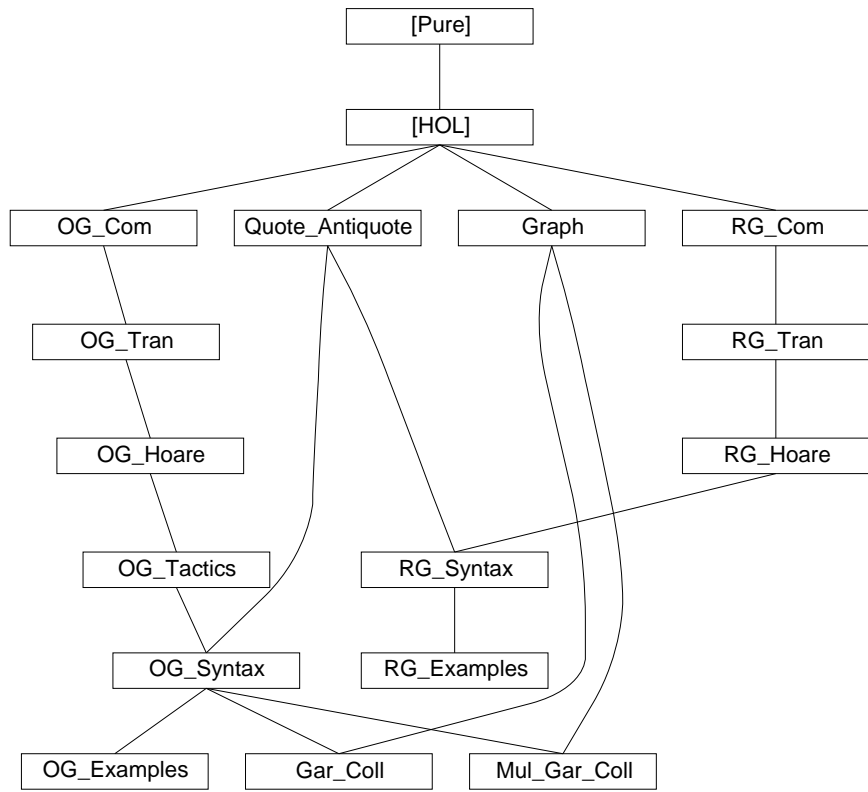
In the following theories a formalization of the Owicki-Gries and the rely-guarantee methods is presented. These methods are widely used for correctness proofs of parallel imperative programs with shared variables. We define syntax, semantics and proof rules in Isabelle/HOL. The proof rules also provide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

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

Contents

1	The Owicki-Gries Method	4
1.1	Abstract Syntax	4
1.2	Operational Semantics	5
1.2.1	The Transition Relation	5
1.2.2	Definition of Semantics	7
1.3	Validity of Correctness Formulas	9
1.4	The Proof System	9
1.5	Soundness	10
1.5.1	Soundness of the System for Atomic Programs	11
1.5.2	Soundness of the System for Component Programs	11
1.5.3	Soundness of the System for Parallel Programs	12
1.6	Generation of Verification Conditions	14
1.7	Concrete Syntax	19
1.8	Examples	21
1.8.1	Mutual Exclusion	21
1.8.2	Parallel Zero Search	24
1.8.3	Producer/Consumer	26
1.8.4	Parameterized Examples	27
2	Case Study: Single and Multi-Mutator Garbage Collection Algorithms	29
2.1	Formalization of the Memory	29
2.1.1	Proofs about Graphs	30
2.2	The Single Mutator Case	32
2.2.1	The Mutator	32
2.2.2	The Collector	33
2.2.3	Interference Freedom	38
2.3	The Multi-Mutator Case	40
2.3.1	The Mutators	41
2.3.2	The Collector	42
2.3.3	Interference Freedom	48

3	The Rely-Guarantee Method	51
3.1	Abstract Syntax	51
3.2	Operational Semantics	51
3.2.1	Semantics of Component Programs	51
3.2.2	Semantics of Parallel Programs	52
3.2.3	Computations	53
3.2.4	Modular Definition of Computation	53
3.2.5	Equivalence of Both Definitions.	54
3.3	Validity of Correctness Formulas	55
3.3.1	Validity for Component Programs.	55
3.3.2	Validity for Parallel Programs.	56
3.3.3	Compositionality of the Semantics	56
3.3.4	The Semantics is Compositional	58
3.4	The Proof System	58
3.4.1	Proof System for Component Programs	59
3.4.2	Proof System for Parallel Programs	60
3.5	Soundness	60
3.5.1	Soundness of the System for Component Programs	62
3.5.2	Soundness of the System for Parallel Programs	64
3.6	Concrete Syntax	66
3.7	Examples	67
3.7.1	Set Elements of an Array to Zero	67
3.7.2	Increment a Variable in Parallel	68
3.7.3	Find Least Element	69



Chapter 1

The Owicki-Gries Method

1.1 Abstract Syntax

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

Type abbreviations for boolean expressions and assertions:

types

'a bexp = *'a set*
'a assn = *'a set*

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

datatype *'a ann-com* =

AnnBasic (*'a assn*) (*'a \Rightarrow 'a*)
| *AnnSeq* (*'a ann-com*) (*'a ann-com*)
| *AnnCond1* (*'a assn*) (*'a bexp*) (*'a ann-com*) (*'a ann-com*)
| *AnnCond2* (*'a assn*) (*'a bexp*) (*'a ann-com*)
| *AnnWhile* (*'a assn*) (*'a bexp*) (*'a assn*) (*'a ann-com*)
| *AnnAwait* (*'a assn*) (*'a bexp*) (*'a com*)

and *'a com* =

Parallel (*'a ann-com option* \times *'a assn*) *list*
| *Basic* (*'a \Rightarrow 'a*)
| *Seq* (*'a com*) (*'a com*)
| *Cond* (*'a bexp*) (*'a com*) (*'a com*)
| *While* (*'a bexp*) (*'a assn*) (*'a com*)

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

consts

pre :: *'a ann-com* \Rightarrow *'a assn*

primrec

pre (*AnnBasic* *r f*) = *r*
pre (*AnnSeq* *c1 c2*) = *pre c1*
pre (*AnnCond1* *r b c1 c2*) = *r*
pre (*AnnCond2* *r b c*) = *r*
pre (*AnnWhile* *r b i c*) = *r*

$pre \ (AnnAwait \ r \ b \ c) = r$

Well-formedness predicate for atomic programs:

```

consts atom-com :: 'a com  $\Rightarrow$  bool
primrec
  atom-com (Parallel Ts) = False
  atom-com (Basic f) = True
  atom-com (Seq c1 c2) = (atom-com c1  $\wedge$  atom-com c2)
  atom-com (Cond b c1 c2) = (atom-com c1  $\wedge$  atom-com c2)
  atom-com (While b i c) = atom-com c

end

```

1.2 Operational Semantics

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

```

types
  'a ann-com-op = ('a ann-com) option
  'a ann-triple-op = ('a ann-com-op  $\times$  'a assn)

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

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

constdefs
  All-None :: 'a ann-triple-op list  $\Rightarrow$  bool
  All-None Ts  $\equiv \forall (c, q) \in \text{set } Ts. c = \text{None}$ 

```

1.2.1 The Transition Relation

```

inductive-set
  ann-transition :: (('a ann-com-op  $\times$  'a)  $\times$  ('a ann-com-op  $\times$  'a)) set
  and transition :: (('a com  $\times$  'a)  $\times$  ('a com  $\times$  'a)) set
  and ann-transition' :: ('a ann-com-op  $\times$  'a)  $\Rightarrow$  ('a ann-com-op  $\times$  'a)  $\Rightarrow$  bool
    (-  $-1 \rightarrow$  -[81,81] 100)
  and transition' :: ('a com  $\times$  'a)  $\Rightarrow$  ('a com  $\times$  'a)  $\Rightarrow$  bool
    (-  $-P1 \rightarrow$  -[81,81] 100)
  and transitions :: ('a com  $\times$  'a)  $\Rightarrow$  ('a com  $\times$  'a)  $\Rightarrow$  bool
    (-  $-P* \rightarrow$  -[81,81] 100)

where
  con-0  $-1 \rightarrow$  con-1  $\equiv (con-0, con-1) \in \text{ann-transition}$ 
  | con-0  $-P1 \rightarrow$  con-1  $\equiv (con-0, con-1) \in \text{transition}$ 
  | con-0  $-P* \rightarrow$  con-1  $\equiv (con-0, con-1) \in \text{transition}^*$ 

  | AnnBasic: (Some (AnnBasic r f), s)  $-1 \rightarrow$  (None, f s)

```

$| \text{AnnSeq1}: (\text{Some } c0, s) -1 \rightarrow (\text{None}, t) \implies$
 $\quad (\text{Some } (\text{AnnSeq } c0 \ c1), s) -1 \rightarrow (\text{Some } c1, t)$
 $| \text{AnnSeq2}: (\text{Some } c0, s) -1 \rightarrow (\text{Some } c2, t) \implies$
 $\quad (\text{Some } (\text{AnnSeq } c0 \ c1), s) -1 \rightarrow (\text{Some } (\text{AnnSeq } c2 \ c1), t)$

 $| \text{AnnCond1T}: s \in b \implies (\text{Some } (\text{AnnCond1 } r \ b \ c1 \ c2), s) -1 \rightarrow (\text{Some } c1, s)$
 $| \text{AnnCond1F}: s \notin b \implies (\text{Some } (\text{AnnCond1 } r \ b \ c1 \ c2), s) -1 \rightarrow (\text{Some } c2, s)$

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

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

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

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

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

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

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

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

monos *rtranscl-mono*

The corresponding syntax translations are:

abbreviation

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

abbreviation

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

abbreviation

$\text{transition-}n :: ('a \ \text{com} \times 'a) \Rightarrow \text{nat} \Rightarrow ('a \ \text{com} \times 'a) \Rightarrow \text{bool}$
 $\quad (- \dashrightarrow \text{--} [81,81,81] \ 100) \ \mathbf{where}$
 $\text{con-}0 \ -Pn \rightarrow \text{con-}1 \equiv (\text{con-}0, \text{con-}1) \in \text{transition}^n$

1.2.2 Definition of Semantics

constdefs

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

$ann\text{-}SEM :: 'a \text{ ann-com} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$
 $ann\text{-}SEM \ c \ S \equiv \bigcup ann\text{-}sem \ c \ ` \ S$

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

$SEM :: 'a \text{ com} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$
 $SEM \ c \ S \equiv \bigcup sem \ c \ ` \ S$

syntax $\text{-}\Omega :: 'a \text{ com} \quad (\Omega \ 63)$

translations $\Omega \Rightarrow While \ UNIV \ UNIV \ (Basic \ id)$

consts $fw\!hile :: 'a \text{ bexp} \Rightarrow 'a \text{ com} \Rightarrow nat \Rightarrow 'a \text{ com}$

primrec

$fw\!hile \ b \ c \ 0 = \Omega$
 $fw\!hile \ b \ c \ (Suc \ n) = Cond \ b \ (Seq \ c \ (fw\!hile \ b \ c \ n)) \ (Basic \ id)$

Proofs

declare $ann\text{-}transition\text{-}transition.intros \ [intro]$

inductive-cases $transition\text{-}cases$:

$(Parallel \ T, s) \dashv\!\!\rightarrow P1 \rightarrow t$
 $(Basic \ f, \ s) \dashv\!\!\rightarrow P1 \rightarrow t$
 $(Seq \ c1 \ c2, \ s) \dashv\!\!\rightarrow P1 \rightarrow t$
 $(Cond \ b \ c1 \ c2, \ s) \dashv\!\!\rightarrow P1 \rightarrow t$
 $(While \ b \ i \ c, \ s) \dashv\!\!\rightarrow P1 \rightarrow t$

lemma $Parallel\text{-}empty\text{-}lemma \ [rule\text{-}format \ (no\text{-}asm)]$:

$(Parallel \ [], s) \dashv\!\!\rightarrow Pn \rightarrow (Parallel \ Ts, t) \longrightarrow Ts = [] \wedge n = 0 \wedge s = t$
 $\langle proof \rangle$

lemma $Parallel\text{-}AllNone\text{-}lemma \ [rule\text{-}format \ (no\text{-}asm)]$:

$All\text{-}None \ Ss \longrightarrow (Parallel \ Ss, s) \dashv\!\!\rightarrow Pn \rightarrow (Parallel \ Ts, t) \longrightarrow Ts = Ss \wedge n = 0 \wedge s = t$
 $\langle proof \rangle$

lemma $Parallel\text{-}AllNone$: $All\text{-}None \ Ts \Longrightarrow (SEM \ (Parallel \ Ts) \ X) = X$

$\langle proof \rangle$

lemma $Parallel\text{-}empty$: $Ts = [] \Longrightarrow (SEM \ (Parallel \ Ts) \ X) = X$

$\langle proof \rangle$

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

lemma $L3\text{-}5i$: $X \subseteq Y \Longrightarrow SEM \ c \ X \subseteq SEM \ c \ Y$

$\langle proof \rangle$

lemma *L3-5ii-lemma1*:

$\llbracket (c1, s1) -P* \rightarrow (Parallel\ Ts, s2); All-None\ Ts;$
 $(c2, s2) -P* \rightarrow (Parallel\ Ss, s3); All-None\ Ss \rrbracket$
 $\implies (Seq\ c1\ c2, s1) -P* \rightarrow (Parallel\ Ss, s3)$
 $\langle proof \rangle$

lemma *L3-5ii-lemma2* [*rule-format* (*no-asm*)]:

$\forall c1\ c2\ s\ t. (Seq\ c1\ c2, s) -Pn \rightarrow (Parallel\ Ts, t) \longrightarrow$
 $(All-None\ Ts) \longrightarrow (\exists y\ m\ Rs. (c1, s) -P* \rightarrow (Parallel\ Rs, y) \wedge$
 $(All-None\ Rs) \wedge (c2, y) -Pm \rightarrow (Parallel\ Ts, t) \wedge m \leq n)$
 $\langle proof \rangle$

lemma *L3-5ii-lemma3*:

$\llbracket (Seq\ c1\ c2, s) -P* \rightarrow (Parallel\ Ts, t); All-None\ Ts \rrbracket \implies$
 $(\exists y\ Rs. (c1, s) -P* \rightarrow (Parallel\ Rs, y) \wedge All-None\ Rs$
 $\wedge (c2, y) -P* \rightarrow (Parallel\ Ts, t))$
 $\langle proof \rangle$

lemma *L3-5ii*: $SEM\ (Seq\ c1\ c2)\ X = SEM\ c2\ (SEM\ c1\ X)$

$\langle proof \rangle$

lemma *L3-5iii*: $SEM\ (Seq\ (Seq\ c1\ c2)\ c3)\ X = SEM\ (Seq\ c1\ (Seq\ c2\ c3))\ X$

$\langle proof \rangle$

lemma *L3-5iv*:

$SEM\ (Cond\ b\ c1\ c2)\ X = (SEM\ c1\ (X \cap b))\ Un\ (SEM\ c2\ (X \cap (-b)))$
 $\langle proof \rangle$

lemma *L3-5v-lemma1* [*rule-format*]:

$(S, s) -Pn \rightarrow (T, t) \longrightarrow S = \Omega \longrightarrow (\neg(\exists Rs. T = (Parallel\ Rs) \wedge All-None\ Rs))$
 $\langle proof \rangle$

lemma *L3-5v-lemma2*: $\llbracket (\Omega, s) -P* \rightarrow (Parallel\ Ts, t); All-None\ Ts \rrbracket \implies False$

$\langle proof \rangle$

lemma *L3-5v-lemma3*: $SEM\ (\Omega)\ S = \{\}$

$\langle proof \rangle$

lemma *L3-5v-lemma4* [*rule-format*]:

$\forall s. (While\ b\ i\ c, s) -Pn \rightarrow (Parallel\ Ts, t) \longrightarrow All-None\ Ts \longrightarrow$
 $(\exists k. (fwhile\ b\ c\ k, s) -P* \rightarrow (Parallel\ Ts, t))$
 $\langle proof \rangle$

lemma *L3-5v-lemma5* [*rule-format*]:

$\forall s. (fwhile\ b\ c\ k, s) -P* \rightarrow (Parallel\ Ts, t) \longrightarrow All-None\ Ts \longrightarrow$
 $(While\ b\ i\ c, s) -P* \rightarrow (Parallel\ Ts, t)$

$\langle proof \rangle$

lemma *L3-5v*: $SEM \ (While \ b \ i \ c) = (\lambda x. (\bigcup k. SEM \ (fwhile \ b \ c \ k) \ x))$
 $\langle proof \rangle$

1.3 Validity of Correctness Formulas

constdefs

com-validity :: $'a \ assn \Rightarrow 'a \ com \Rightarrow 'a \ assn \Rightarrow bool$ $((\exists || = -// -// -) \ [90,55,90]$
 $50)$

$|| = p \ c \ q \equiv SEM \ c \ p \subseteq q$

ann-com-validity :: $'a \ ann-com \Rightarrow 'a \ assn \Rightarrow bool$ $(| = - - \ [60,90] \ 45)$

$| = c \ q \equiv ann-SEM \ c \ (pre \ c) \subseteq q$

end

1.4 The Proof System

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

consts *assertions* :: $'a \ ann-com \Rightarrow ('a \ assn) \ set$

primrec

assertions $(AnnBasic \ r \ f) = \{r\}$

assertions $(AnnSeq \ c1 \ c2) = assertions \ c1 \cup assertions \ c2$

assertions $(AnnCond1 \ r \ b \ c1 \ c2) = \{r\} \cup assertions \ c1 \cup assertions \ c2$

assertions $(AnnCond2 \ r \ b \ c) = \{r\} \cup assertions \ c$

assertions $(AnnWhile \ r \ b \ i \ c) = \{r, i\} \cup assertions \ c$

assertions $(AnnAwait \ r \ b \ c) = \{r\}$

consts *atomics* :: $'a \ ann-com \Rightarrow ('a \ assn \times 'a \ com) \ set$

primrec

atomics $(AnnBasic \ r \ f) = \{(r, Basic \ f)\}$

atomics $(AnnSeq \ c1 \ c2) = atomics \ c1 \cup atomics \ c2$

atomics $(AnnCond1 \ r \ b \ c1 \ c2) = atomics \ c1 \cup atomics \ c2$

atomics $(AnnCond2 \ r \ b \ c) = atomics \ c$

atomics $(AnnWhile \ r \ b \ i \ c) = atomics \ c$

atomics $(AnnAwait \ r \ b \ c) = \{(r \cap b, c)\}$

consts *com* :: $'a \ ann-triple-op \Rightarrow 'a \ ann-com-op$

primrec *com* $(c, q) = c$

consts *post* :: $'a \ ann-triple-op \Rightarrow 'a \ assn$

primrec *post* $(c, q) = q$

constdefs *interfree-aux* :: $('a \ ann-com-op \times 'a \ assn \times 'a \ ann-com-op) \Rightarrow bool$

interfree-aux $\equiv \lambda (co, q, co'). \ co' = None \vee$

$(\forall (r, a) \in atomics \ (the \ co')). \ || = (q \cap r) \ a \ q \wedge$

$$(co = None \vee (\forall p \in \text{assertions } (the\ co). \models (p \cap r) \ a\ p)))$$

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

inductive

oghoare :: $\text{'a assn} \Rightarrow \text{'a com} \Rightarrow \text{'a assn} \Rightarrow bool$ $((\exists \parallel - \text{ } - // - // -) [90,55,90] 50)$
and *ann-hoare* :: $\text{'a ann-com} \Rightarrow \text{'a assn} \Rightarrow bool$ $((2 \vdash - // -) [60,90] 45)$

where

AnnBasic: $r \subseteq \{s. f\ s \in q\} \Longrightarrow \vdash (AnnBasic\ r\ f)\ q$

| *AnnSeq*: $\llbracket \vdash\ c0\ pre\ c1; \vdash\ c1\ q \rrbracket \Longrightarrow \vdash (AnnSeq\ c0\ c1)\ q$

| *AnnCond1*: $\llbracket r \cap b \subseteq pre\ c1; \vdash\ c1\ q; r \cap -b \subseteq pre\ c2; \vdash\ c2\ q \rrbracket$
 $\Longrightarrow \vdash (AnnCond1\ r\ b\ c1\ c2)\ q$

| *AnnCond2*: $\llbracket r \cap b \subseteq pre\ c; \vdash\ c\ q; r \cap -b \subseteq q \rrbracket \Longrightarrow \vdash (AnnCond2\ r\ b\ c)\ q$

| *AnnWhile*: $\llbracket r \subseteq i; i \cap b \subseteq pre\ c; \vdash\ c\ i; i \cap -b \subseteq q \rrbracket$
 $\Longrightarrow \vdash (AnnWhile\ r\ b\ i\ c)\ q$

| *AnnAwait*: $\llbracket atom-com\ c; \parallel - (r \cap b)\ c\ q \rrbracket \Longrightarrow \vdash (AnnAwait\ r\ b\ c)\ q$

| *AnnConseq*: $\llbracket \vdash\ c\ q; q \subseteq q' \rrbracket \Longrightarrow \vdash\ c\ q'$

| *Parallel*: $\llbracket \forall i < \text{length } Ts. \exists c\ q. Ts!i = (Some\ c, q) \wedge \vdash\ c\ q; interfree\ Ts \rrbracket$
 $\Longrightarrow \parallel - (\bigcap_{i \in \{i. i < \text{length } Ts\}}. pre(the(com(Ts!i))))$
Parallel *Ts*
 $(\bigcap_{i \in \{i. i < \text{length } Ts\}}. post(Ts!i))$

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

| *Seq*: $\llbracket \parallel -\ p\ c1\ r; \parallel -\ r\ c2\ q \rrbracket \Longrightarrow \parallel -\ p\ (Seq\ c1\ c2)\ q$

| *Cond*: $\llbracket \parallel -\ (p \cap b)\ c1\ q; \parallel -\ (p \cap -b)\ c2\ q \rrbracket \Longrightarrow \parallel -\ p\ (Cond\ b\ c1\ c2)\ q$

| *While*: $\llbracket \parallel -\ (p \cap b)\ c\ p \rrbracket \Longrightarrow \parallel -\ p\ (While\ b\ i\ c)\ (p \cap -b)$

| *Conseq*: $\llbracket p' \subseteq p; \parallel -\ p\ c\ q; q \subseteq q' \rrbracket \Longrightarrow \parallel -\ p'\ c\ q'$

1.5 Soundness

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

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

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

lemmas *AnnBasic* = *oghoare-ann-hoare.AnnBasic*

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

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

1.5.1 Soundness of the System for Atomic Programs

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

lemma $SEM\text{-}fwhile$: $SEM\ S\ (p \cap b) \subseteq p \implies SEM\ (fwhile\ b\ S\ k)\ p \subseteq (p \cap \neg b)$
 $\langle proof \rangle$

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

1.5.2 Soundness of the System for Component Programs

inductive-cases $ann\text{-}transition\text{-}cases$:

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

Strong Soundness for Component Programs:

lemma $ann\text{-}hoare\text{-}case\text{-}analysis$ [rule-format]:
defines I : $I \equiv \lambda C\ q'.$
 $((\forall r\ f. C = AnnBasic\ r\ f \longrightarrow (\exists q. r \subseteq \{s. f\ s \in q\} \wedge q \subseteq q')) \wedge$
 $(\forall c0\ c1. C = AnnSeq\ c0\ c1 \longrightarrow (\exists q. q \subseteq q' \wedge \vdash c0\ pre\ c1 \wedge \vdash c1\ q)) \wedge$
 $(\forall r\ b\ c1\ c2. C = AnnCond1\ r\ b\ c1\ c2 \longrightarrow (\exists q. q \subseteq q' \wedge$
 $r \cap b \subseteq pre\ c1 \wedge \vdash c1\ q \wedge r \cap \neg b \subseteq pre\ c2 \wedge \vdash c2\ q)) \wedge$
 $(\forall r\ b\ c. C = AnnCond2\ r\ b\ c \longrightarrow$
 $(\exists q. q \subseteq q' \wedge r \cap b \subseteq pre\ c \wedge \vdash c\ q \wedge r \cap \neg b \subseteq q)) \wedge$
 $(\forall r\ i\ b\ c. C = AnnWhile\ r\ b\ i\ c \longrightarrow$
 $(\exists q. q \subseteq q' \wedge r \subseteq i \wedge i \cap b \subseteq pre\ c \wedge \vdash c\ i \wedge i \cap \neg b \subseteq q)) \wedge$

$(\forall r \ b \ c. \ C = \text{AnnAwait } r \ b \ c \longrightarrow (\exists q. \ q \subseteq q' \wedge \Vdash (r \cap b) \ c \ q)))$
shows $\vdash C \ q' \longrightarrow I \ C \ q'$
 $\langle \text{proof} \rangle$

lemma *Help*: $(\text{transition} \cap \{(x,y). \ \text{True}\}) = (\text{transition})$
 $\langle \text{proof} \rangle$

lemma *Strong-Soundness-aux-aux* [rule-format]:
 $(co, s) -1 \rightarrow (co', t) \longrightarrow (\forall c. \ co = \text{Some } c \longrightarrow s \in \text{pre } c \longrightarrow$
 $(\forall q. \ \vdash c \ q \longrightarrow (\text{if } co' = \text{None} \text{ then } t \in q \text{ else } t \in \text{pre}(\text{the } co') \wedge \vdash (\text{the } co') \ q)))$
 $\langle \text{proof} \rangle$

lemma *Strong-Soundness-aux*: $\llbracket (\text{Some } c, s) -*\rightarrow (co, t); s \in \text{pre } c; \vdash c \ q \rrbracket$
 $\implies \text{if } co = \text{None} \text{ then } t \in q \text{ else } t \in \text{pre}(\text{the } co) \wedge \vdash (\text{the } co) \ q$
 $\langle \text{proof} \rangle$

lemma *Strong-Soundness*: $\llbracket (\text{Some } c, s) -*\rightarrow (co, t); s \in \text{pre } c; \vdash c \ q \rrbracket$
 $\implies \text{if } co = \text{None} \text{ then } t \in q \text{ else } t \in \text{pre}(\text{the } co)$
 $\langle \text{proof} \rangle$

lemma *ann-hoare-sound*: $\vdash c \ q \implies \models c \ q$
 $\langle \text{proof} \rangle$

1.5.3 Soundness of the System for Parallel Programs

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

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

lemma *assertions-lemma*: $\text{pre } c \in \text{assertions } c$
 $\langle \text{proof} \rangle$

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

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

lemma *interfree-lemma*: $\llbracket (\text{Some } c, s) -1 \rightarrow (r, t); \text{interfree } Ts ; i < \text{length } Ts;$
 $Ts!i = (\text{Some } c, q) \rrbracket \implies \text{interfree } (Ts[i := (r, q)])$
 $\langle \text{proof} \rangle$

Strong Soundness Theorem for Parallel Programs:

lemma *Parallel-Strong-Soundness-Seq-aux*:

$\llbracket \text{interfree } Ts; i < \text{length } Ts; \text{com}(Ts ! i) = \text{Some}(\text{AnnSeq } c0 \ c1) \rrbracket$
 $\implies \text{interfree } (Ts[i := (\text{Some } c0, \text{pre } c1)])$
 $\langle \text{proof} \rangle$

lemma *Parallel-Strong-Soundness-Seq [rule-format (no-asm)]*:

$\llbracket \forall i < \text{length } Ts. (\text{if } \text{com}(Ts ! i) = \text{None} \text{ then } b \in \text{post}(Ts ! i)$
 $\text{ else } b \in \text{pre}(\text{the}(\text{com}(Ts ! i))) \wedge \vdash \text{the}(\text{com}(Ts ! i)) \text{ post}(Ts ! i);$
 $\text{com}(Ts ! i) = \text{Some}(\text{AnnSeq } c0 \ c1); i < \text{length } Ts; \text{interfree } Ts \rrbracket \implies$
 $(\forall ia < \text{length } Ts. (\text{if } \text{com}(Ts[i := (\text{Some } c0, \text{pre } c1)] ! ia) = \text{None}$
 $\text{ then } b \in \text{post}(Ts[i := (\text{Some } c0, \text{pre } c1)] ! ia)$
 $\text{ else } b \in \text{pre}(\text{the}(\text{com}(Ts[i := (\text{Some } c0, \text{pre } c1)] ! ia))) \wedge$
 $\vdash \text{the}(\text{com}(Ts[i := (\text{Some } c0, \text{pre } c1)] ! ia)) \text{ post}(Ts[i := (\text{Some } c0, \text{pre } c1)] ! ia)))$
 $\wedge \text{interfree } (Ts[i := (\text{Some } c0, \text{pre } c1)])$
 $\langle \text{proof} \rangle$

lemma *Parallel-Strong-Soundness-aux-aux [rule-format]*:

$(\text{Some } c, b) - 1 \rightarrow (co, t) \longrightarrow$
 $(\forall Ts. i < \text{length } Ts \longrightarrow \text{com}(Ts ! i) = \text{Some } c \longrightarrow$
 $(\forall i < \text{length } Ts. (\text{if } \text{com}(Ts ! i) = \text{None} \text{ then } b \in \text{post}(Ts ! i)$
 $\text{ else } b \in \text{pre}(\text{the}(\text{com}(Ts ! i))) \wedge \vdash \text{the}(\text{com}(Ts ! i)) \text{ post}(Ts ! i))) \longrightarrow$
 $\text{interfree } Ts \longrightarrow$
 $(\forall j. j < \text{length } Ts \wedge i \neq j \longrightarrow (\text{if } \text{com}(Ts ! j) = \text{None} \text{ then } t \in \text{post}(Ts ! j)$
 $\text{ else } t \in \text{pre}(\text{the}(\text{com}(Ts ! j))) \wedge \vdash \text{the}(\text{com}(Ts ! j)) \text{ post}(Ts ! j)))$
 $\langle \text{proof} \rangle$

lemma *Parallel-Strong-Soundness-aux [rule-format]*:

$\llbracket (Ts', s) - P* \rightarrow (Rs', t); Ts' = (\text{Parallel } Ts); \text{interfree } Ts;$
 $\forall i. i < \text{length } Ts \longrightarrow (\exists c \ q. (Ts ! i) = (\text{Some } c, q) \wedge s \in \text{pre } c \wedge \vdash c \ q) \rrbracket \implies$
 $\forall Rs. Rs' = (\text{Parallel } Rs) \longrightarrow (\forall j. j < \text{length } Rs \longrightarrow$
 $(\text{if } \text{com}(Rs ! j) = \text{None} \text{ then } t \in \text{post}(Ts ! j)$
 $\text{ else } t \in \text{pre}(\text{the}(\text{com}(Rs ! j))) \wedge \vdash \text{the}(\text{com}(Rs ! j)) \text{ post}(Ts ! j))) \wedge \text{interfree } Rs$
 $\langle \text{proof} \rangle$

lemma *Parallel-Strong-Soundness*:

$\llbracket (\text{Parallel } Ts, s) - P* \rightarrow (\text{Parallel } Rs, t); \text{interfree } Ts; j < \text{length } Rs;$
 $\forall i. i < \text{length } Ts \longrightarrow (\exists c \ q. Ts ! i = (\text{Some } c, q) \wedge s \in \text{pre } c \wedge \vdash c \ q) \rrbracket \implies$
 $\text{if } \text{com}(Rs ! j) = \text{None} \text{ then } t \in \text{post}(Ts ! j) \text{ else } t \in \text{pre}(\text{the}(\text{com}(Rs ! j)))$
 $\langle \text{proof} \rangle$

lemma *oghoare-sound [rule-format]*: $\llbracket - \ p \ c \ q \longrightarrow \rrbracket = p \ c \ q$

$\langle \text{proof} \rangle$

end

1.6 Generation of Verification Conditions

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

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

lemmas *oghoare-intros*=*Parallel Basic Seq Cond While Conseq*

lemma *ParallelConseqRule*:

$$\begin{aligned} & \llbracket p \subseteq (\bigcap i \in \{i. i < \text{length } Ts\}. \text{pre}(\text{the}(\text{com}(Ts ! i))))); \\ & \quad \llbracket - (\bigcap i \in \{i. i < \text{length } Ts\}. \text{pre}(\text{the}(\text{com}(Ts ! i)))) \\ & \quad \quad (\text{Parallel } Ts) \\ & \quad (\bigcap i \in \{i. i < \text{length } Ts\}. \text{post}(Ts ! i)); \\ & \quad (\bigcap i \in \{i. i < \text{length } Ts\}. \text{post}(Ts ! i)) \subseteq q \rrbracket \\ & \implies \llbracket - p (\text{Parallel } Ts) q \rrbracket \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *SkipRule*: $p \subseteq q \implies \llbracket - p (\text{Basic id}) q \rrbracket$
 $\langle \text{proof} \rangle$

lemma *BasicRule*: $p \subseteq \{s. (f s) \in q\} \implies \llbracket - p (\text{Basic } f) q \rrbracket$
 $\langle \text{proof} \rangle$

lemma *SeqRule*: $\llbracket \llbracket - p \text{ c1 } r; \llbracket - r \text{ c2 } q \rrbracket \rrbracket \implies \llbracket - p (\text{Seq c1 c2}) q \rrbracket$
 $\langle \text{proof} \rangle$

lemma *CondRule*:

$$\begin{aligned} & \llbracket p \subseteq \{s. (s \in b \longrightarrow s \in w) \wedge (s \notin b \longrightarrow s \in w')\}; \llbracket - w \text{ c1 } q; \llbracket - w' \text{ c2 } q \rrbracket \rrbracket \\ & \implies \llbracket - p (\text{Cond } b \text{ c1 c2}) q \rrbracket \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *WhileRule*: $\llbracket p \subseteq i; \llbracket - (i \cap b) \text{ c } i; (i \cap (-b)) \subseteq q \rrbracket \implies \llbracket - p (\text{While } b \text{ i c}) q \rrbracket$
 $\langle \text{proof} \rangle$

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

lemma *AnnatomRule*:

$$\llbracket \text{atom-com}(c); \llbracket - r \text{ c } q \rrbracket \rrbracket \implies \vdash (\text{AnnAwait } r \{s. \text{True}\} c) q$$

 $\langle \text{proof} \rangle$

lemma *AnnskipRule*:

$$r \subseteq q \implies \vdash (\text{AnnBasic } r \text{ id}) q$$

 $\langle \text{proof} \rangle$

lemma *AnnwaitRule*:

$\llbracket (r \cap b) \subseteq q \rrbracket \implies \vdash (\text{AnnAwait } r \ b \ (\text{Basic id})) \ q$
 $\langle \text{proof} \rangle$

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

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

lemma *interfree-aux-rule2*:
 $\forall (R, r) \in (\text{atomics } a). \llbracket - \ (q \cap R) \ r \ q \implies \text{interfree-aux}(\text{None}, q, \text{Some } a)$
 $\langle \text{proof} \rangle$

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

lemma *AnnBasic-assertions*:
 $\llbracket \text{interfree-aux}(\text{None}, r, \text{Some } a); \text{interfree-aux}(\text{None}, q, \text{Some } a) \rrbracket \implies$
 $\text{interfree-aux}(\text{Some } (\text{AnnBasic } r \ f), q, \text{Some } a)$
 $\langle \text{proof} \rangle$

lemma *AnnSeq-assertions*:
 $\llbracket \text{interfree-aux}(\text{Some } c1, q, \text{Some } a); \text{interfree-aux}(\text{Some } c2, q, \text{Some } a) \rrbracket \implies$
 $\text{interfree-aux}(\text{Some } (\text{AnnSeq } c1 \ c2), q, \text{Some } a)$
 $\langle \text{proof} \rangle$

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

lemma *AnnCond2-assertions*:
 $\llbracket \text{interfree-aux}(\text{None}, r, \text{Some } a); \text{interfree-aux}(\text{Some } c, q, \text{Some } a) \rrbracket \implies$
 $\text{interfree-aux}(\text{Some } (\text{AnnCond2 } r \ b \ c), q, \text{Some } a)$
 $\langle \text{proof} \rangle$

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

lemma *AnnAwait-assertions*:
 $\llbracket \text{interfree-aux}(\text{None}, r, \text{Some } a); \text{interfree-aux}(\text{None}, q, \text{Some } a) \rrbracket \implies$
 $\text{interfree-aux}(\text{Some } (\text{AnnAwait } r \ b \ c), q, \text{Some } a)$
 $\langle \text{proof} \rangle$

lemma *AnnBasic-atomics*:

$\llbracket - \rrbracket (q \cap r) \text{ (Basic } f) \ q \implies \text{interfree-aux}(\text{None}, q, \text{Some } (\text{AnnBasic } r \ f))$
 $\langle \text{proof} \rangle$

lemma *AnnSeq-atomics*:

$\llbracket \text{interfree-aux}(\text{Any}, q, \text{Some } a1); \text{interfree-aux}(\text{Any}, q, \text{Some } a2) \rrbracket \implies$
 $\text{interfree-aux}(\text{Any}, q, \text{Some } (\text{AnnSeq } a1 \ a2))$
 $\langle \text{proof} \rangle$

lemma *AnnCond1-atomics*:

$\llbracket \text{interfree-aux}(\text{Any}, q, \text{Some } a1); \text{interfree-aux}(\text{Any}, q, \text{Some } a2) \rrbracket \implies$
 $\text{interfree-aux}(\text{Any}, q, \text{Some } (\text{AnnCond1 } r \ b \ a1 \ a2))$
 $\langle \text{proof} \rangle$

lemma *AnnCond2-atomics*:

$\text{interfree-aux}(\text{Any}, q, \text{Some } a) \implies \text{interfree-aux}(\text{Any}, q, \text{Some } (\text{AnnCond2 } r \ b \ a))$
 $\langle \text{proof} \rangle$

lemma *AnnWhile-atomics*: $\text{interfree-aux}(\text{Any}, q, \text{Some } a)$

$\implies \text{interfree-aux}(\text{Any}, q, \text{Some } (\text{AnnWhile } r \ b \ i \ a))$
 $\langle \text{proof} \rangle$

lemma *Annatom-atomics*:

$\llbracket - \rrbracket (q \cap r) \ a \ q \implies \text{interfree-aux}(\text{None}, q, \text{Some } (\text{AnnAwait } r \ \{x. \text{True}\} \ a))$
 $\langle \text{proof} \rangle$

lemma *AnnAwait-atomics*:

$\llbracket - \rrbracket (q \cap (r \cap b)) \ a \ q \implies \text{interfree-aux}(\text{None}, q, \text{Some } (\text{AnnAwait } r \ b \ a))$
 $\langle \text{proof} \rangle$

constdefs

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

lemma *interfree-swap-Empty*: $\text{interfree-swap}(x, [])$

$\langle \text{proof} \rangle$

lemma *interfree-swap-List*:

$\llbracket \text{interfree-aux}(\text{com } x, \text{post } x, \text{com } y);$
 $\text{interfree-aux}(\text{com } y, \text{post } y, \text{com } x); \text{interfree-swap}(x, xs) \rrbracket$
 $\implies \text{interfree-swap}(x, y \# xs)$
 $\langle \text{proof} \rangle$

lemma *interfree-swap-Map*: $\forall k. i \leq k \wedge k < j \longrightarrow \text{interfree-aux}(\text{com } x, \text{post } x, c$
 $k)$

$\wedge \text{interfree-aux}(c \ k, Q \ k, \text{com } x)$
 $\implies \text{interfree-swap}(x, \text{map } (\lambda k. (c \ k, Q \ k)) [i..<j])$

$\langle proof \rangle$

lemma *interfree-Empty*: *interfree* []
 $\langle proof \rangle$

lemma *interfree-List*:
 $\llbracket \text{interfree-swap}(x, xs); \text{interfree } xs \rrbracket \implies \text{interfree } (x \# xs)$
 $\langle proof \rangle$

lemma *interfree-Map*:
 $(\forall i j. a \leq i \wedge i < b \wedge a \leq j \wedge j < b \wedge i \neq j \longrightarrow \text{interfree-aux } (c \ i, Q \ i, c \ j))$
 $\implies \text{interfree } (\text{map } (\lambda k. (c \ k, Q \ k)) [a..<b])$
 $\langle proof \rangle$

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

lemma *MapAnnEmpty*: $\llbracket \cdot \rrbracket []$
 $\langle proof \rangle$

lemma *MapAnnList*: $\llbracket \vdash c \ q ; \llbracket \cdot \rrbracket xs \rrbracket \implies \llbracket \cdot \rrbracket (\text{Some } c, q) \# xs$
 $\langle proof \rangle$

lemma *MapAnnMap*:
 $\forall k. i \leq k \wedge k < j \longrightarrow \vdash (c \ k) (Q \ k) \implies \llbracket \cdot \rrbracket \text{map } (\lambda k. (\text{Some } (c \ k), Q \ k)) [i..<j]$
 $\langle proof \rangle$

lemma *ParallelRule*: $\llbracket \llbracket \cdot \rrbracket Ts ; \text{interfree } Ts \rrbracket$
 $\implies \llbracket - (\bigcap i \in \{i. i < \text{length } Ts\}. \text{pre}(\text{the}(\text{com}(Ts!i))))$
 $\text{Parallel } Ts$
 $(\bigcap i \in \{i. i < \text{length } Ts\}. \text{post}(Ts!i))$
 $\langle proof \rangle$

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

lemma *Compl-Collect*: $\neg(\text{Collect } b) = \{x. \neg(b \ x)\}$
 $\langle proof \rangle$

lemma *list-length*: $\text{length } [] = 0 \wedge \text{length } (x \# xs) = \text{Suc}(\text{length } xs)$
 $\langle proof \rangle$

lemma *list-lemmas*: $\text{length } [] = 0 \wedge \text{length } (x \# xs) = \text{Suc}(\text{length } xs)$
 $\wedge (x \# xs) ! 0 = x \wedge (x \# xs) ! \text{Suc } n = xs ! n$
 $\langle proof \rangle$

lemma *le-Suc-eq-insert*: $\{i. i < \text{Suc } n\} = \text{insert } n \ \{i. i < n\}$
 $\langle proof \rangle$

lemmas *primrecdef-list* = *pre.simps assertions.simps atomics.simps atom-com.simps*

lemmas *my-simp-list* = *list-lemmas fst-conv snd-conv*
not-less0 refl le-Suc-eq-insert Suc-not-Zero Zero-not-Suc Suc-Suc-eq
Collect-mem-eq ball-simps option.simps primrecdef-list

lemmas *ParallelConseq-list = INTER-def Collect-conj-eq length-map length-upt length-append list-length*

⟨ML⟩

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

⟨ML⟩

Tactic for the generation of the verification conditions

The tactic basically uses two subtactics:

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

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

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

⟨ML⟩

The final tactic is given the name *oghoare*:

⟨ML⟩

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

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

⟨ML⟩

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

⟨ML⟩

Tactics useful for dealing with the generated verification conditions:

⟨ML⟩

end

1.7 Concrete Syntax

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

syntax

-quote :: 'b \Rightarrow ('a \Rightarrow 'b) ((\ll - \gg) [0] 1000)
 -antiquote :: ('a \Rightarrow 'b) \Rightarrow 'b ('- [1000] 1000)
 -Assert :: 'a \Rightarrow 'a set ((.{-}.) [0] 1000)

syntax (*xsymbols*)

-Assert :: 'a \Rightarrow 'a set (($\{\}$ - $\}$) [0] 1000)

translations

.{b}. \rightarrow *Collect* $\ll b \gg$

$\langle ML \rangle$

end

theory *OG-Syntax*

imports *OG-Tactics* *Quote-Antiquote*

begin

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

syntax

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

translations

' *x* := *a* \rightarrow *Basic* \ll ' (-update-name *x* (λ -. *a*)) \gg
r ' *x* := *a* \rightarrow *AnnBasic* *r* \ll ' (-update-name *x* (λ -. *a*)) \gg

syntax

-AnnSkip :: 'a *assn* \Rightarrow 'a *ann-com* (-//SKIP [90] 63)
 -AnnSeq :: 'a *ann-com* \Rightarrow 'a *ann-com* \Rightarrow 'a *ann-com* (-;;/ - [60,61] 60)
 -AnnCond1 :: 'a *assn* \Rightarrow 'a *bexp* \Rightarrow 'a *ann-com* \Rightarrow 'a *ann-com* \Rightarrow 'a *ann-com*
 (- //IF - /THEN - /ELSE - /FI [90,0,0,0] 61)
 -AnnCond2 :: 'a *assn* \Rightarrow 'a *bexp* \Rightarrow 'a *ann-com* \Rightarrow 'a *ann-com*
 (- //IF - /THEN - /FI [90,0,0] 61)
 -AnnWhile :: 'a *assn* \Rightarrow 'a *bexp* \Rightarrow 'a *assn* \Rightarrow 'a *ann-com* \Rightarrow 'a *ann-com*
 (- //WHILE - /INV - //DO -//OD [90,0,0,0] 61)
 -AnnAwait :: 'a *assn* \Rightarrow 'a *bexp* \Rightarrow 'a *com* \Rightarrow 'a *ann-com*
 (- //AWAIT - /THEN /- /END [90,0,0] 61)
 -AnnAtom :: 'a *assn* \Rightarrow 'a *com* \Rightarrow 'a *ann-com* (-//<-> [90,0] 61)

-AnnWait :: 'a assn \Rightarrow 'a bexp \Rightarrow 'a ann-com (-// WAIT - END [90,0] 61)
-Skip :: 'a com (SKIP 63)
-Seq :: 'a com \Rightarrow 'a com \Rightarrow 'a com (-,/- [55, 56] 55)
-Cond :: 'a bexp \Rightarrow 'a com \Rightarrow 'a com \Rightarrow 'a com
((0IF -/ THEN -/ ELSE -/ FI) [0, 0, 0] 61)
-Cond2 :: 'a bexp \Rightarrow 'a com \Rightarrow 'a com (IF - THEN - FI [0,0] 56)
-While-inv :: 'a bexp \Rightarrow 'a assn \Rightarrow 'a com \Rightarrow 'a com
((0WHILE -/ INV - //DO - /OD) [0, 0, 0] 61)
-While :: 'a bexp \Rightarrow 'a com \Rightarrow 'a com
((0WHILE - //DO - /OD) [0, 0] 61)

translations

SKIP \rightleftharpoons Basic id

c-1, *c-2* \rightleftharpoons Seq *c-1* *c-2*

IF b THEN c1 ELSE c2 FI \rightarrow Cond .{b}. *c1 c2*

IF b THEN c FI \rightleftharpoons *IF b THEN c ELSE SKIP FI*

WHILE b INV i DO c OD \rightarrow While .{b}. *i c*

WHILE b DO c OD \rightleftharpoons *WHILE b INV arbitrary DO c OD*

r SKIP \rightleftharpoons AnnBasic *r id*

c-1; *c-2* \rightleftharpoons AnnSeq *c-1* *c-2*

r IF b THEN c1 ELSE c2 FI \rightarrow AnnCond1 *r* .{b}. *c1 c2*

r IF b THEN c FI \rightarrow AnnCond2 *r* .{b}. *c*

r WHILE b INV i DO c OD \rightarrow AnnWhile *r* .{b}. *i c*

r AWAIT b THEN c END \rightarrow AnnAwait *r* .{b}. *c*

r $\langle c \rangle$ \rightleftharpoons *r AWAIT True THEN c END*

r WAIT b END \rightleftharpoons *r AWAIT b THEN SKIP END*

nonterminals

prgs

syntax

-PAR :: *prgs* \Rightarrow 'a (COBEGIN -// -// COEND [57] 56)

-prg :: ['a, 'a] \Rightarrow *prgs* (-// - [60, 90] 57)

-prgs :: ['a, 'a, *prgs*] \Rightarrow *prgs* (-// -// - [60,90,57] 57)

-prg-scheme :: ['a, 'a, 'a, 'a, 'a] \Rightarrow *prgs*

(SCHEME [- \leq - < -] -// - [0,0,0,60, 90] 57)

translations

-prg c q \rightleftharpoons [(Some *c*, *q*)]

-prgs c q ps \rightleftharpoons (Some *c*, *q*) # *ps*

-PAR ps \rightleftharpoons Parallel *ps*

-prg-scheme j i k c q \rightleftharpoons map (λi . (Some *c*, *q*)) [*j*..*k*]

$\langle ML \rangle$

end

1.8 Examples

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

1.8.1 Mutual Exclusion

Peterson's Algorithm I

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

```
record Petersons-mutex-1 =
  pr1 :: nat
  pr2 :: nat
  in1 :: bool
  in2 :: bool
  hold :: nat

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

Peterson's Algorithm II: A Busy Wait Solution

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

```
record Busy-wait-mutex =
  flag1 :: bool
  flag2 :: bool
  turn :: nat
  after1 :: bool
  after2 :: bool
```

lemma *Busy-wait-mutex*:

```
||- .{True}.
  'flag1:=False,, 'flag2:=False,,
  COBEGIN .{¬'flag1}.
    WHILE True
    INV .{¬'flag1}.
    DO .{¬'flag1}. { 'flag1:=True,, 'after1:=False };;
      .{ 'flag1 ∧ ¬'after1 }. { 'turn:=1,, 'after1:=True };;
      .{ 'flag1 ∧ 'after1 ∧ ('turn=1 ∨ 'turn=2) }.
        WHILE ¬('flag2 → 'turn=2)
        INV .{ 'flag1 ∧ 'after1 ∧ ('turn=1 ∨ 'turn=2) }.
        DO .{ 'flag1 ∧ 'after1 ∧ ('turn=1 ∨ 'turn=2) }. SKIP OD;;
      .{ 'flag1 ∧ 'after1 ∧ ('flag2 ∧ 'after2 → 'turn=2) }.
      'flag1:=False
    OD
  .{False}.
||
  .{¬'flag2}.
  WHILE True
  INV .{¬'flag2}.
  DO .{¬'flag2}. { 'flag2:=True,, 'after2:=False };;
    .{ 'flag2 ∧ ¬'after2 }. { 'turn:=2,, 'after2:=True };;
    .{ 'flag2 ∧ 'after2 ∧ ('turn=1 ∨ 'turn=2) }.
      WHILE ¬('flag1 → 'turn=1)
      INV .{ 'flag2 ∧ 'after2 ∧ ('turn=1 ∨ 'turn=2) }.
      DO .{ 'flag2 ∧ 'after2 ∧ ('turn=1 ∨ 'turn=2) }. SKIP OD;;
    .{ 'flag2 ∧ 'after2 ∧ ('flag1 ∧ 'after1 → 'turn=1) }.
    'flag2:=False
  OD
  .{False}.
COEND
.{False}.
⟨proof⟩
```

Peterson's Algorithm III: A Solution using Semaphores

```
record Semaphores-mutex =
  out :: bool
```


who :: *nat*

lemma *Semaphores-mutex*:

```

||- .{i≠j}.
  'out:=True ,,
  COBEGIN .{i≠j}.
    WHILE True INV .{i≠j}.
    DO .{i≠j}. AWAIT 'out THEN 'out:=False,, 'who:=i END;;
    .{¬'out ∧ 'who=i ∧ i≠j}. 'out:=True OD
    .{False}.
  ||
    .{i≠j}.
    WHILE True INV .{i≠j}.
    DO .{i≠j}. AWAIT 'out THEN 'out:=False,, 'who:=j END;;
    .{¬'out ∧ 'who=j ∧ i≠j}. 'out:=True OD
    .{False}.
  COEND
  .{False}.
⟨proof⟩

```

Peterson's Algorithm III: Parameterized version:

lemma *Semaphores-parameterized-mutex*:

```

0 < n ⇒ ||- .{True}.
  'out:=True ,,
  COBEGIN
    SCHEME [0 ≤ i < n]
    .{True}.
    WHILE True INV .{True}.
    DO .{True}. AWAIT 'out THEN 'out:=False,, 'who:=i END;;
    .{¬'out ∧ 'who=i}. 'out:=True OD
    .{False}.
  COEND
  .{False}.
⟨proof⟩

```

The Ticket Algorithm

record *Ticket-mutex* =

```

num :: nat
nextv :: nat
turn :: nat list
index :: nat

```

lemma *Ticket-mutex*:

```

[[ 0 < n; I = <<n=length 'turn ∧ 0 < 'nextv ∧ (∀ k l. k < n ∧ l < n ∧ k ≠ l
  ⇒ 'turn!k < 'num ∧ ('turn!k = 0 ∨ 'turn!k ≠ 'turn!l))>> ]]
⇒ ||- .{n=length 'turn}.
  'index:= 0,,
  WHILE 'index < n INV .{n=length 'turn ∧ (∀ i < 'index. 'turn!i=0)}.

```

```

DO 'turn:= 'turn['index:=0],, 'index:='index +1 OD,,
'num:=1 ,, 'nextv:=1 ,,
COBEGIN
SCHEME [0 ≤ i < n]
.{ 'I}.
WHILE True INV .{ 'I}.
DO .{ 'I}. < 'turn := 'turn[i:= 'num],, 'num:= 'num+1 >;
.{ 'I}. WAIT 'turn!i= 'nextv END;;
.{ 'I ∧ 'turn!i= 'nextv}. 'nextv:= 'nextv+1
OD
.{ False}.
COEND
.{ False}.
<proof>

```

1.8.2 Parallel Zero Search

Synchronized Zero Search. Zero-6

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

record Zero-search =

```

turn :: nat
found :: bool
x :: nat
y :: nat

```

lemma Zero-search:

```

[[I1 = < a ≤ 'x ∧ ('found → (a < 'x ∧ f('x)=0) ∨ ('y ≤ a ∧ f('y)=0))
  ∧ (¬ 'found ∧ a < 'x → f('x) ≠ 0) > ;
I2 = < 'y ≤ a+1 ∧ ('found → (a < 'x ∧ f('x)=0) ∨ ('y ≤ a ∧ f('y)=0))
  ∧ (¬ 'found ∧ 'y ≤ a → f('y) ≠ 0) > ]] ⇒
||- .{ ∃ u. f(u)=0 }.
'turn:=1,, 'found:= False,,
'x:=a,, 'y:=a+1 ,,
COBEGIN .{ 'I1}.
WHILE ¬ 'found
INV .{ 'I1}.
DO .{ a ≤ 'x ∧ ('found → 'y ≤ a ∧ f('y)=0) ∧ (a < 'x → f('x) ≠ 0) }.
WAIT 'turn=1 END;;
.{ a ≤ 'x ∧ ('found → 'y ≤ a ∧ f('y)=0) ∧ (a < 'x → f('x) ≠ 0) }.
'turn:=2;;
.{ a ≤ 'x ∧ ('found → 'y ≤ a ∧ f('y)=0) ∧ (a < 'x → f('x) ≠ 0) }.
< 'x:= 'x+1,,
IF f('x)=0 THEN 'found:=True ELSE SKIP FI>
OD;;
.{ 'I1 ∧ 'found }.
'turn:=2
.{ 'I1 ∧ 'found }.

```

```

||
.{ 'I2}.
  WHILE  $\neg$ 'found
  INV .{'I2}.
  DO .{'y ≤ a+1 ∧ ('found → a < 'x ∧ f('x)=0) ∧ ('y ≤ a → f('y) ≠ 0)}.
    WAIT 'turn=2 END;;
    .{'y ≤ a+1 ∧ ('found → a < 'x ∧ f('x)=0) ∧ ('y ≤ a → f('y) ≠ 0)}.
    'turn:=1;;
    .{'y ≤ a+1 ∧ ('found → a < 'x ∧ f('x)=0) ∧ ('y ≤ a → f('y) ≠ 0)}.
    ⟨ 'y:=( 'y - 1),,
      IF f('y)=0 THEN 'found:=True ELSE SKIP FI⟩
    OD;;
    .{'I2 ∧ 'found}.
    'turn:=1
    .{'I2 ∧ 'found}.
  COEND
  .{f('x)=0 ∨ f('y)=0}.
⟨proof⟩

```

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

lemma Zero-Search-2:

```

[[I1 = << a ≤ 'x ∧ ('found → (a < 'x ∧ f('x)=0) ∨ ('y ≤ a ∧ f('y)=0))
  ∧ (¬'found ∧ a < 'x → f('x) ≠ 0)>>;
I2 = << 'y ≤ a+1 ∧ ('found → (a < 'x ∧ f('x)=0) ∨ ('y ≤ a ∧ f('y)=0))
  ∧ (¬'found ∧ 'y ≤ a → f('y) ≠ 0)>>]] ⇒
|| - .{∃ u. f(u)=0}.
  'found:= False,,
  'x:=a,, 'y:=a+1,,
  COBEGIN .{'I1}.
    WHILE  $\neg$ 'found
    INV .{'I1}.
    DO .{a ≤ 'x ∧ ('found → 'y ≤ a ∧ f('y)=0) ∧ (a < 'x → f('x) ≠ 0)}.
      ⟨ 'x:='x+1,, IF f('x)=0 THEN 'found:=True ELSE SKIP FI⟩
    OD
    .{'I1 ∧ 'found}.
  ||
  .{'I2}.
  WHILE  $\neg$ 'found
  INV .{'I2}.
  DO .{'y ≤ a+1 ∧ ('found → a < 'x ∧ f('x)=0) ∧ ('y ≤ a → f('y) ≠ 0)}.
    ⟨ 'y:=( 'y - 1),, IF f('y)=0 THEN 'found:=True ELSE SKIP FI⟩
  OD
  .{'I2 ∧ 'found}.
  COEND
  .{f('x)=0 ∨ f('y)=0}.
⟨proof⟩

```

1.8.3 Producer/Consumer

Previous lemmas

lemma *nat-lemma2*: $\llbracket b = m*(n::nat) + t; a = s*n + u; t=u; b-a < n \rrbracket \implies m \leq s$
<proof>

lemma *mod-lemma*: $\llbracket (c::nat) \leq a; a < b; b - c < n \rrbracket \implies b \bmod n \neq a \bmod n$
<proof>

Producer/Consumer Algorithm

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

The whole proof takes aprox. 4 minutes.

lemma *Producer-consumer*:

$\llbracket INIT = \llbracket 0 < length\ a \wedge 0 < length\ 'buffer \wedge length\ 'b = length\ a \rrbracket ;$
 $I = \llbracket (\forall k < 'ins. 'outs \leq k \longrightarrow (a ! k) = 'buffer ! (k \bmod (length\ 'buffer))) \wedge$
 $'outs \leq 'ins \wedge 'ins - 'outs \leq length\ 'buffer \rrbracket ;$
 $I1 = \llbracket I \wedge 'li \leq length\ a \rrbracket ;$
 $p1 = \llbracket I1 \wedge 'li = 'ins \rrbracket ;$
 $I2 = \llbracket I \wedge (\forall k < 'lj. (a ! k) = ('b ! k)) \wedge 'lj \leq length\ a \rrbracket ;$
 $p2 = \llbracket I2 \wedge 'lj = 'outs \rrbracket \rrbracket \implies$
 $\llbracket - .\{ 'INIT \}.$
 $'ins := 0,, 'outs := 0,, 'li := 0,, 'lj := 0,,$
 $COBEGIN .\{ 'p1 \wedge 'INIT \}.$
 $WHILE 'li < length\ a$
 $INV .\{ 'p1 \wedge 'INIT \}.$
 $DO .\{ 'p1 \wedge 'INIT \wedge 'li < length\ a \}.$
 $'vx := (a ! 'li);;$
 $.\{ 'p1 \wedge 'INIT \wedge 'li < length\ a \wedge 'vx = (a ! 'li) \}.$
 $WAIT 'ins - 'outs < length\ 'buffer\ END;;$
 $.\{ 'p1 \wedge 'INIT \wedge 'li < length\ a \wedge 'vx = (a ! 'li)$
 $\wedge 'ins - 'outs < length\ 'buffer \}.$
 $'buffer := (list-update\ 'buffer\ ('ins \bmod (length\ 'buffer))\ 'vx);;$
 $.\{ 'p1 \wedge 'INIT \wedge 'li < length\ a$
 $\wedge (a ! 'li) = ('buffer ! ('ins \bmod (length\ 'buffer)))$
 $\wedge 'ins - 'outs < length\ 'buffer \}.$
 $'ins := 'ins + 1;;$
 $.\{ 'I1 \wedge 'INIT \wedge ('li + 1) = 'ins \wedge 'li < length\ a \}.$
 $'li := 'li + 1$

```

OD
.{ 'p1 ∧ 'INIT ∧ 'li=length a }.
||
.{ 'p2 ∧ 'INIT }.
WHILE 'lj < length a
  INV .{ 'p2 ∧ 'INIT }.
DO .{ 'p2 ∧ 'lj < length a ∧ 'INIT }.
  WAIT 'outs < 'ins END;;
  .{ 'p2 ∧ 'lj < length a ∧ 'outs < 'ins ∧ 'INIT }.
  'vy := ('buffer ! ('outs mod (length 'buffer)));;
  .{ 'p2 ∧ 'lj < length a ∧ 'outs < 'ins ∧ 'vy = (a ! 'lj) ∧ 'INIT }.
  'outs := 'outs + 1;;
  .{ 'I2 ∧ ('lj + 1) = 'outs ∧ 'lj < length a ∧ 'vy = (a ! 'lj) ∧ 'INIT }.
  'b := (list-update 'b 'lj 'vy);;
  .{ 'I2 ∧ ('lj + 1) = 'outs ∧ 'lj < length a ∧ (a ! 'lj) = ('b ! 'lj) ∧ 'INIT }.
  'lj := 'lj + 1
OD
.{ 'p2 ∧ 'lj = length a ∧ 'INIT }.
COEND
.{ ∀ k < length a. (a ! k) = ('b ! k) }.
⟨proof⟩

```

1.8.4 Parameterized Examples

Set Elements of an Array to Zero

```

record Example1 =
  a :: nat ⇒ nat

```

lemma Example1:

```

||- .{ True }.
COBEGIN SCHEME [0 ≤ i < n] .{ True }. 'a := 'a (i := 0) .{ 'a i = 0 }. COEND
.{ ∀ i < n. 'a i = 0 }.
⟨proof⟩

```

Same example with lists as auxiliary variables.

```

record Example1-list =
  A :: nat list

```

lemma Example1-list:

```

||- .{ n < length 'A }.
COBEGIN
  SCHEME [0 ≤ i < n] .{ n < length 'A }. 'A := 'A[i := 0] .{ 'A i = 0 }.
COEND
.{ ∀ i < n. 'A i = 0 }.
⟨proof⟩

```

Increment a Variable in Parallel

First some lemmas about summation properties.

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

lemma *Example2-lemma2-aux2*:
 $!!b. j \leq s \implies (\sum_{i::nat=0..<j.} (b \ (s:=t)) \ i) = (\sum_{i=0..<j.} b \ i)$
 $\langle proof \rangle$

lemma *Example2-lemma2*:
 $!!b. \llbracket j < n; \ b \ j = 0 \rrbracket \implies Suc \ (\sum_{i::nat=0..<n.} b \ i) = (\sum_{i=0..<n.} (b \ (j := Suc \ 0)) \ i)$
 $\langle proof \rangle$

record *Example2* =
 $c :: nat \Rightarrow nat$
 $x :: nat$

lemma *Example-2*: $0 < n \implies$
 $\| - . \{ 'x = 0 \wedge (\sum_{i=0..<n.} 'c \ i) = 0 \}.$
 $COBEGIN$
 $\quad SCHEME \ [0 \leq i < n]$
 $\quad . \{ 'x = (\sum_{i=0..<n.} 'c \ i) \wedge 'c \ i = 0 \}.$
 $\quad \langle 'x := 'x + (Suc \ 0),, 'c := 'c \ (i := (Suc \ 0)) \rangle$
 $\quad . \{ 'x = (\sum_{i=0..<n.} 'c \ i) \wedge 'c \ i = (Suc \ 0) \}.$
 $COEND$
 $\quad . \{ 'x = n \}.$
 $\langle proof \rangle$

end

Chapter 2

Case Study: Single and Multi-Mutator Garbage Collection Algorithms

2.1 Formalization of the Memory

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

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

types

nodes = *node list*

edge = $\text{nat} \times \text{nat}$

edges = *edge list*

consts *Roots* :: *nat set*

constdefs

Proper-Roots :: *nodes* \Rightarrow *bool*

Proper-Roots *M* $\equiv \text{Roots} \neq \{\} \wedge \text{Roots} \subseteq \{i. i < \text{length } M\}$

Proper-Edges :: (*nodes* \times *edges*) \Rightarrow *bool*

Proper-Edges $\equiv (\lambda(M, E). \forall i < \text{length } E. \text{fst}(E!i) < \text{length } M \wedge \text{snd}(E!i) < \text{length } M)$

BtoW :: (*edge* \times *nodes*) \Rightarrow *bool*

BtoW $\equiv (\lambda(e, M). (M! \text{fst } e) = \text{Black} \wedge (M! \text{snd } e) \neq \text{Black})$

Blacks :: *nodes* \Rightarrow *nat set*

Blacks *M* $\equiv \{i. i < \text{length } M \wedge M!i = \text{Black}\}$

Reach :: *edges* \Rightarrow *nat set*

Reach *E* $\equiv \{x. (\exists \text{path}. 1 < \text{length } \text{path} \wedge \text{path}!(\text{length } \text{path} - 1) \in \text{Roots} \wedge x = \text{path}!0)$

$$\wedge (\forall i < \text{length } \text{path} - 1. (\exists j < \text{length } E. E!j = (\text{path}!(i+1), \text{path}!i))) \\ \vee x \in \text{Roots}\}$$

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

2.1.1 Proofs about Graphs

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

Graph 1

lemma *Graph1-aux* [rule-format]:

$$\llbracket \text{Roots} \subseteq \text{Blacks } M; \forall i < \text{length } E. \neg \text{BtoW}(E!i, M) \rrbracket \\ \implies 1 < \text{length } \text{path} \longrightarrow (\text{path}!(\text{length } \text{path} - 1)) \in \text{Roots} \longrightarrow \\ (\forall i < \text{length } \text{path} - 1. (\exists j. j < \text{length } E \wedge E!j = (\text{path}!(\text{Suc } i), \text{path}!i))) \\ \longrightarrow M!(\text{path}!0) = \text{Black}$$
 $\langle \text{proof} \rangle$

lemma *Graph1*:

$$\llbracket \text{Roots} \subseteq \text{Blacks } M; \text{Proper-Edges}(M, E); \forall i < \text{length } E. \neg \text{BtoW}(E!i, M) \rrbracket \\ \implies \text{Reach } E \subseteq \text{Blacks } M$$
 $\langle \text{proof} \rangle$

Graph 2

lemma *Ex-first-occurrence* [rule-format]:

$$P (n :: \text{nat}) \longrightarrow (\exists m. P m \wedge (\forall i. i < m \longrightarrow \neg P i))$$
 $\langle \text{proof} \rangle$

lemma *Compl-lemma*: $(n :: \text{nat}) \leq l \implies (\exists m. m \leq l \wedge n = l - m)$
 $\langle \text{proof} \rangle$

lemma *Ex-last-occurrence*:

$$\llbracket P (n :: \text{nat}); n \leq l \rrbracket \implies (\exists m. P (l - m) \wedge (\forall i. i < m \longrightarrow \neg P (l - i)))$$
 $\langle \text{proof} \rangle$

lemma *Graph2*:

$$\llbracket T \in \text{Reach } E; R < \text{length } E \rrbracket \implies T \in \text{Reach } (E[R := (\text{fst}(E!R), T)])$$
 $\langle \text{proof} \rangle$

Graph 3

lemma *Graph3*:

$$\llbracket T \in \text{Reach } E; R < \text{length } E \rrbracket \implies \text{Reach}(E[R := (\text{fst}(E!R), T)]) \subseteq \text{Reach } E$$
 $\langle \text{proof} \rangle$

Graph 4

lemma *Graph4*:

$$\begin{aligned} & \llbracket T \in \text{Reach } E; \text{Roots} \subseteq \text{Blacks } M; I \leq \text{length } E; T < \text{length } M; R < \text{length } E; \\ & \forall i < I. \neg \text{BtoW}(E!i, M); R < I; M!fst(E!R) = \text{Black}; M!T \neq \text{Black} \rrbracket \implies \\ & (\exists r. I \leq r \wedge r < \text{length } E \wedge \text{BtoW}(E[R := (fst(E!R), T)]!r, M)) \\ & \langle \text{proof} \rangle \end{aligned}$$

Graph 5

lemma *Graph5*:

$$\begin{aligned} & \llbracket T \in \text{Reach } E; \text{Roots} \subseteq \text{Blacks } M; \forall i < R. \neg \text{BtoW}(E!i, M); T < \text{length } M; \\ & R < \text{length } E; M!fst(E!R) = \text{Black}; M!snd(E!R) = \text{Black}; M!T \neq \text{Black} \rrbracket \\ & \implies (\exists r. R < r \wedge r < \text{length } E \wedge \text{BtoW}(E[R := (fst(E!R), T)]!r, M)) \\ & \langle \text{proof} \rangle \end{aligned}$$

Other lemmas about graphs

lemma *Graph6*:

$$\llbracket \text{Proper-Edges}(M, E); R < \text{length } E; T < \text{length } M \rrbracket \implies \text{Proper-Edges}(M, E[R := (fst(E!R), T)])$$

 $\langle \text{proof} \rangle$

lemma *Graph7*:

$$\llbracket \text{Proper-Edges}(M, E) \rrbracket \implies \text{Proper-Edges}(M[T := a], E)$$

 $\langle \text{proof} \rangle$

lemma *Graph8*:

$$\llbracket \text{Proper-Roots}(M) \rrbracket \implies \text{Proper-Roots}(M[T := a])$$

 $\langle \text{proof} \rangle$

Some specific lemmata for the verification of garbage collection algorithms.

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

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

lemma *Graph11* [*rule-format* (*no-asm*)]:

$$\llbracket M!j \neq \text{Black}; j < \text{length } M \rrbracket \implies \text{Blacks } M \subset \text{Blacks } (M[j := \text{Black}])$$

 $\langle \text{proof} \rangle$

lemma *Graph12*: $\llbracket a \subseteq \text{Blacks } M; j < \text{length } M \rrbracket \implies a \subseteq \text{Blacks } (M[j := \text{Black}])$
 $\langle \text{proof} \rangle$

lemma *Graph13*: $\llbracket a \subset \text{Blacks } M; j < \text{length } M \rrbracket \implies a \subset \text{Blacks } (M[j := \text{Black}])$
 $\langle \text{proof} \rangle$

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

end

2.2 The Single Mutator Case

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

declare *psubsetE* [*rule del*]

Declaration of variables:

record *gar-coll-state* =

M :: *nodes*
E :: *edges*
bc :: *nat set*
obc :: *nat set*
Ma :: *nodes*
ind :: *nat*
k :: *nat*
z :: *bool*

2.2.1 The Mutator

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

We declare the arbitrarily selected node and edge as constants:

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

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

constdefs

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

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

constdefs

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

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

Mutator :: *gar-coll-state* *ann-com*
Mutator \equiv
 $\{ 'Mut-init \wedge 'z \}.$

$WHILE \text{ True } INV .\{ 'Mut-init \wedge 'z \}.$
 $DO \text{ Redirect-Edge } ;; \text{ Color-Target } OD$

Correctness of the mutator

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

lemma *Redirect-Edge*:
 $\vdash \text{Redirect-Edge } pre(\text{Color-Target})$
 $\langle proof \rangle$

lemma *Color-Target*:
 $\vdash \text{Color-Target} .\{ 'Mut-init \wedge 'z \}.$
 $\langle proof \rangle$

lemma *Mutator*:
 $\vdash \text{Mutator} .\{ False \}.$
 $\langle proof \rangle$

2.2.2 The Collector

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

consts *M-init* :: *nodes*

constdefs

$\text{Proper-M-init} :: \text{gar-coll-state} \Rightarrow \text{bool}$
 $\text{Proper-M-init} \equiv \ll \text{Blacks } M\text{-init} = \text{Roots} \wedge \text{length } M\text{-init} = \text{length } 'M \gg$

$\text{Proper} :: \text{gar-coll-state} \Rightarrow \text{bool}$
 $\text{Proper} \equiv \ll \text{Proper-Roots } 'M \wedge \text{Proper-Edges}('M, 'E) \wedge 'Proper\text{-M-init} \gg$

$\text{Safe} :: \text{gar-coll-state} \Rightarrow \text{bool}$
 $\text{Safe} \equiv \ll \text{Reach } 'E \subseteq \text{Blacks } 'M \gg$

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

Blackening the roots

constdefs

$\text{Blacken-Roots} :: \text{gar-coll-state} \text{ ann-com}$
 $\text{Blacken-Roots} \equiv$
 $.\{ 'Proper \}.$
 $'ind := 0;;$
 $.\{ 'Proper \wedge 'ind = 0 \}.$
 $WHILE 'ind < \text{length } 'M$
 $INV .\{ 'Proper \wedge (\forall i < 'ind. i \in \text{Roots} \longrightarrow 'M[i] = \text{Black}) \wedge 'ind \leq \text{length } 'M \}.$
 $DO .\{ 'Proper \wedge (\forall i < 'ind. i \in \text{Roots} \longrightarrow 'M[i] = \text{Black}) \wedge 'ind < \text{length } 'M \}.$
 $IF 'ind \in \text{Roots} THEN$

$\{ 'Proper \wedge (\forall i < 'ind. i \in Roots \longrightarrow 'M!i=Black) \wedge 'ind < length\ 'M \wedge 'ind \in Roots \}.$
 $'M := 'M['ind := Black] \text{ FI};$
 $\{ 'Proper \wedge (\forall i < 'ind + 1. i \in Roots \longrightarrow 'M!i=Black) \wedge 'ind < length\ 'M \}.$
 $'ind := 'ind + 1$
OD

lemma *Blacken-Roots*:

$\vdash \text{Blacken-Roots } \{ 'Proper \wedge Roots \subseteq Blacks\ 'M \}.$
 $\langle \text{proof} \rangle$

Propagating black

constdefs

$PBInv :: \text{gar-coll-state} \Rightarrow \text{nat} \Rightarrow \text{bool}$
 $PBInv \equiv \ll \lambda ind. 'obc < Blacks\ 'M \vee (\forall i < ind. \neg BtoW ('E!i, 'M) \vee$
 $(\neg z \wedge i = R \wedge (snd('E!R)) = T \wedge (\exists r. ind \leq r \wedge r < length\ 'E \wedge BtoW('E!r, 'M)))) \gg$

constdefs

$Propagate-Black-aux :: \text{gar-coll-state} \text{ ann-com}$
 $Propagate-Black-aux \equiv$
 $\{ 'Proper \wedge Roots \subseteq Blacks\ 'M \wedge 'obc \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M \}.$
 $'ind := 0;$
 $\{ 'Proper \wedge Roots \subseteq Blacks\ 'M \wedge 'obc \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M \wedge 'ind = 0 \}.$

$WHILE\ 'ind < length\ 'E$
 $INV\ \{ 'Proper \wedge Roots \subseteq Blacks\ 'M \wedge 'obc \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge 'PBInv\ 'ind \wedge 'ind \leq length\ 'E \}.$
 $DO\ \{ 'Proper \wedge Roots \subseteq Blacks\ 'M \wedge 'obc \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge 'PBInv\ 'ind \wedge 'ind < length\ 'E \}.$
 $IF\ 'M!(fst('E!'ind)) = Black\ THEN$
 $\{ 'Proper \wedge Roots \subseteq Blacks\ 'M \wedge 'obc \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge 'PBInv\ 'ind \wedge 'ind < length\ 'E \wedge 'M!fst('E!'ind) = Black \}.$
 $'M := 'M[snd('E!'ind) := Black];$
 $\{ 'Proper \wedge Roots \subseteq Blacks\ 'M \wedge 'obc \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge 'PBInv\ ('ind + 1) \wedge 'ind < length\ 'E \}.$
 $'ind := 'ind + 1$
FI
OD

lemma *Propagate-Black-aux*:

$\vdash \text{Propagate-Black-aux}$
 $\{ 'Proper \wedge Roots \subseteq Blacks\ 'M \wedge 'obc \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge ('obc < Blacks\ 'M \vee 'Safe) \}.$
 $\langle \text{proof} \rangle$

Refining propagating black

constdefs

$Auxk :: \text{gar-coll-state} \Rightarrow \text{bool}$

$$\begin{aligned} Auxk \equiv & \ll 'k < \text{length } 'M \wedge ('M! 'k \neq \text{Black} \vee \neg BtoW('E! 'ind, 'M) \vee \\ & 'obc < \text{Blacks } 'M \vee (\neg 'z \wedge 'ind = R \wedge \text{snd}('E! R) = T \\ & \wedge (\exists r. 'ind < r \wedge r < \text{length } 'E \wedge BtoW('E! r, 'M))) \gg \end{aligned}$$

constdefs

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

Propagate-Black \equiv

.{ 'Proper \wedge Roots \subseteq Blacks 'M \wedge 'obc \subseteq Blacks 'M \wedge 'bc \subseteq Blacks 'M }.
'ind := 0;;

.{ 'Proper \wedge Roots \subseteq Blacks 'M \wedge 'obc \subseteq Blacks 'M \wedge 'bc \subseteq Blacks 'M \wedge 'ind = 0 }.

WHILE 'ind < length 'E

INV .{ 'Proper \wedge Roots \subseteq Blacks 'M \wedge 'obc \subseteq Blacks 'M \wedge 'bc \subseteq Blacks 'M
 \wedge 'PBIInv 'ind \wedge 'ind \leq length 'E }.

DO .{ 'Proper \wedge Roots \subseteq Blacks 'M \wedge 'obc \subseteq Blacks 'M \wedge 'bc \subseteq Blacks 'M
 \wedge 'PBIInv 'ind \wedge 'ind < length 'E }.

IF ('M!(fst ('E! 'ind))) = Black THEN

.{ 'Proper \wedge Roots \subseteq Blacks 'M \wedge 'obc \subseteq Blacks 'M \wedge 'bc \subseteq Blacks 'M
 \wedge 'PBIInv 'ind \wedge 'ind < length 'E \wedge ('M!fst('E! 'ind)) = Black }.
'k := (snd('E! 'ind));;

.{ 'Proper \wedge Roots \subseteq Blacks 'M \wedge 'obc \subseteq Blacks 'M \wedge 'bc \subseteq Blacks 'M
 \wedge 'PBIInv 'ind \wedge 'ind < length 'E \wedge ('M!fst('E! 'ind)) = Black
 \wedge 'Auxk }.

< 'M := 'M['k := Black],, 'ind := 'ind + 1 >

ELSE .{ 'Proper \wedge Roots \subseteq Blacks 'M \wedge 'obc \subseteq Blacks 'M \wedge 'bc \subseteq Blacks 'M
 \wedge 'PBIInv 'ind \wedge 'ind < length 'E }.

< IF ('M!(fst ('E! 'ind))) \neq Black THEN 'ind := 'ind + 1 FI >

FI

OD

lemma *Propagate-Black*:

\vdash *Propagate-Black*

.{ 'Proper \wedge Roots \subseteq Blacks 'M \wedge 'obc \subseteq Blacks 'M \wedge 'bc \subseteq Blacks 'M
 \wedge ('obc < Blacks 'M \vee 'Safe) }.

<proof>

Counting black nodes

constdefs

CountInv :: *gar-coll-state \Rightarrow nat \Rightarrow bool*

CountInv $\equiv \ll \lambda ind. \{i. i < ind \wedge 'Ma!i = \text{Black}\} \subseteq 'bc \gg$

constdefs

Count :: *gar-coll-state ann-com*

Count \equiv

.{ 'Proper \wedge Roots \subseteq Blacks 'M
 \wedge 'obc \subseteq Blacks 'Ma \wedge Blacks 'Ma \subseteq Blacks 'M \wedge 'bc \subseteq Blacks 'M
 \wedge length 'Ma = length 'M \wedge ('obc < Blacks 'Ma \vee 'Safe) \wedge 'bc = {} }.

'ind := 0;;

.{ 'Proper \wedge Roots \subseteq Blacks 'M

$\wedge 'obc \subseteq \text{Blacks } 'Ma \wedge \text{Blacks } 'Ma \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$
 $\wedge \text{length } 'Ma = \text{length } 'M \wedge ('obc < \text{Blacks } 'Ma \vee 'Safe) \wedge 'bc = \{\}$
 $\wedge 'ind = 0\}$.
WHILE $'ind < \text{length } 'M$
 INV $\{ 'Proper \wedge \text{Roots} \subseteq \text{Blacks } 'M$
 $\wedge 'obc \subseteq \text{Blacks } 'Ma \wedge \text{Blacks } 'Ma \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$
 $\wedge \text{length } 'Ma = \text{length } 'M \wedge 'CountInv 'ind$
 $\wedge ('obc < \text{Blacks } 'Ma \vee 'Safe) \wedge 'ind \leq \text{length } 'M\}$.
 DO $\{ 'Proper \wedge \text{Roots} \subseteq \text{Blacks } 'M$
 $\wedge 'obc \subseteq \text{Blacks } 'Ma \wedge \text{Blacks } 'Ma \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$
 $\wedge \text{length } 'Ma = \text{length } 'M \wedge 'CountInv 'ind$
 $\wedge ('obc < \text{Blacks } 'Ma \vee 'Safe) \wedge 'ind < \text{length } 'M\}$.
 IF $'M! 'ind = \text{Black}$
 THEN $\{ 'Proper \wedge \text{Roots} \subseteq \text{Blacks } 'M$
 $\wedge 'obc \subseteq \text{Blacks } 'Ma \wedge \text{Blacks } 'Ma \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$
 $\wedge \text{length } 'Ma = \text{length } 'M \wedge 'CountInv 'ind$
 $\wedge ('obc < \text{Blacks } 'Ma \vee 'Safe) \wedge 'ind < \text{length } 'M \wedge 'M! 'ind = \text{Black}\}$.
 $'bc := \text{insert } 'ind 'bc$
 FI;;
 $\{ 'Proper \wedge \text{Roots} \subseteq \text{Blacks } 'M$
 $\wedge 'obc \subseteq \text{Blacks } 'Ma \wedge \text{Blacks } 'Ma \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$
 $\wedge \text{length } 'Ma = \text{length } 'M \wedge 'CountInv ('ind + 1)$
 $\wedge ('obc < \text{Blacks } 'Ma \vee 'Safe) \wedge 'ind < \text{length } 'M\}$.
 $'ind := 'ind + 1$
OD

lemma *Count*:

$\vdash \text{Count}$
 $\{ 'Proper \wedge \text{Roots} \subseteq \text{Blacks } 'M$
 $\wedge 'obc \subseteq \text{Blacks } 'Ma \wedge \text{Blacks } 'Ma \subseteq 'bc \wedge 'bc \subseteq \text{Blacks } 'M \wedge \text{length } 'Ma = \text{length } 'M$
 $\wedge ('obc < \text{Blacks } 'Ma \vee 'Safe)\}$.
<proof>

Appending garbage nodes to the free list

consts *Append-to-free* :: $\text{nat} \times \text{edges} \Rightarrow \text{edges}$

axioms

Append-to-free0: $\text{length } (\text{Append-to-free } (i, e)) = \text{length } e$
Append-to-free1: $\text{Proper-Edges } (m, e)$
 $\implies \text{Proper-Edges } (m, \text{Append-to-free}(i, e))$
Append-to-free2: $i \notin \text{Reach } e$
 $\implies n \in \text{Reach } (\text{Append-to-free}(i, e)) = (n = i \vee n \in \text{Reach } e)$

constdefs

AppendInv :: $\text{gar-coll-state} \Rightarrow \text{nat} \Rightarrow \text{bool}$
AppendInv $\equiv \ll \lambda ind. \forall i < \text{length } 'M. ind \leq i \longrightarrow i \in \text{Reach } 'E \longrightarrow 'M!i = \text{Black} \gg$

constdefs

```

Append :: gar-coll-state ann-com
Append ≡
.{ 'Proper ∧ Roots ⊆ Blacks 'M ∧ 'Safe}.
'ind := 0;;
.{ 'Proper ∧ Roots ⊆ Blacks 'M ∧ 'Safe ∧ 'ind = 0}.
WHILE 'ind < length 'M
  INV .{ 'Proper ∧ 'AppendInv 'ind ∧ 'ind ≤ length 'M}.
DO .{ 'Proper ∧ 'AppendInv 'ind ∧ 'ind < length 'M}.
  IF 'M! 'ind = Black THEN
    .{ 'Proper ∧ 'AppendInv 'ind ∧ 'ind < length 'M ∧ 'M! 'ind = Black}.
    'M := 'M[ 'ind := White]
  ELSE .{ 'Proper ∧ 'AppendInv 'ind ∧ 'ind < length 'M ∧ 'ind ∉ Reach 'E}.
    'E := Append-to-free('ind, 'E)
  FI;;
.{ 'Proper ∧ 'AppendInv ('ind + 1) ∧ 'ind < length 'M}.
'ind := 'ind + 1
OD

```

lemma Append:

$\vdash \text{Append} .\{ 'Proper \}.$
 $\langle \text{proof} \rangle$

Correctness of the Collector**constdefs**

```

Collector :: gar-coll-state ann-com
Collector ≡
.{ 'Proper}.
WHILE True INV .{ 'Proper}.
DO
  Blacken-Roots;;
  .{ 'Proper ∧ Roots ⊆ Blacks 'M}.
  'obc := {};
  .{ 'Proper ∧ Roots ⊆ Blacks 'M ∧ 'obc = {}}.
  'bc := Roots;;
  .{ 'Proper ∧ Roots ⊆ Blacks 'M ∧ 'obc = {} ∧ 'bc = Roots}.
  'Ma := M-init;;
  .{ 'Proper ∧ Roots ⊆ Blacks 'M ∧ 'obc = {} ∧ 'bc = Roots ∧ 'Ma = M-init}.
  WHILE 'obc ≠ 'bc
    INV .{ 'Proper ∧ Roots ⊆ Blacks 'M
      ∧ 'obc ⊆ Blacks 'Ma ∧ Blacks 'Ma ⊆ 'bc ∧ 'bc ⊆ Blacks 'M
      ∧ length 'Ma = length 'M ∧ ('obc < Blacks 'Ma ∨ 'Safe)}.
    DO .{ 'Proper ∧ Roots ⊆ Blacks 'M ∧ 'bc ⊆ Blacks 'M}.
      'obc := 'bc;;
      Propagate-Black;;
      .{ 'Proper ∧ Roots ⊆ Blacks 'M ∧ 'obc ⊆ Blacks 'M ∧ 'bc ⊆ Blacks 'M
        ∧ ('obc < Blacks 'M ∨ 'Safe)}.
      'Ma := 'M;;

```

```

    .{ ' Proper  $\wedge$  Roots $\subseteq$ Blacks ' M  $\wedge$  ' obc $\subseteq$ Blacks ' Ma
       $\wedge$  Blacks ' Ma $\subseteq$ Blacks ' M  $\wedge$  ' bc $\subseteq$ Blacks ' M  $\wedge$  length ' Ma=length ' M
       $\wedge$  ( ' obc < Blacks ' Ma  $\vee$  ' Safe)}.
    ' bc:={};;
    Count
  OD;;
  Append
  OD

```

lemma *Collector*:
 \vdash Collector .{False}.
 <proof>

2.2.3 Interference Freedom

lemmas *modules* = Redirect-Edge-def Color-Target-def Blacken-Roots-def
 Propagate-Black-def Count-def Append-def
lemmas *Invariants* = PBIInv-def Auxk-def CountInv-def AppendInv-def
lemmas *abbrev* = collector-defs mutator-defs Invariants

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

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

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

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

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

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

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

lemma *interfree-Color-Target-Propagate-Black*:

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

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

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

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

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

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

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

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

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

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

Interference freedom Collector-Mutator

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

$\langle proof \rangle$

Interference freedom Mutator-Collector

lemma *interfree-Mutator-Collector*:
interfree-aux (Some Mutator, {}, Some Collector)
 $\langle proof \rangle$

The Garbage Collection algorithm

In total there are 289 verification conditions.

lemma *Gar-Coll*:
|| $\vdash \{ \text{'Proper} \wedge \text{'Mut-init} \wedge \text{'z} \}.$
COBEGIN
Collector
. $\{False\}.$
||
Mutator
. $\{False\}.$
COEND
. $\{False\}.$
 $\langle proof \rangle$
end

2.3 The Multi-Mutator Case

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

The full theory takes aprox. 18 minutes.

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

Declaration of variables:

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

2.3.1 The Mutators

constdefs

Mul-mut-init :: *mul-gar-coll-state* \Rightarrow *nat* \Rightarrow *bool*
Mul-mut-init $\equiv \ll \lambda n. n = \text{length } 'Muts \wedge (\forall i < n. R ('Muts!i) < \text{length } 'E$
 $\wedge T ('Muts!i) < \text{length } 'M) \gg$

Mul-Redirect-Edge :: *nat* \Rightarrow *nat* \Rightarrow *mul-gar-coll-state* *ann-com*

Mul-Redirect-Edge *j n* \equiv

$\{ 'Mul-mut-init\ n \wedge Z ('Muts!j) \}$.

$\langle IF\ T ('Muts!j) \in Reach\ 'E\ THEN$

$'E := 'E[R ('Muts!j) := (fst ('E!R ('Muts!j)), T ('Muts!j))] FI,,$

$'Muts := 'Muts[j := ('Muts!j) (Z := False)] \rangle$

Mul-Color-Target :: *nat* \Rightarrow *nat* \Rightarrow *mul-gar-coll-state* *ann-com*

Mul-Color-Target *j n* \equiv

$\{ 'Mul-mut-init\ n \wedge \neg Z ('Muts!j) \}$.

$\langle 'M := 'M[T ('Muts!j) := Black],, 'Muts := 'Muts[j := ('Muts!j) (Z := True)] \rangle$

Mul-Mutator :: *nat* \Rightarrow *nat* \Rightarrow *mul-gar-coll-state* *ann-com*

Mul-Mutator *j n* \equiv

$\{ 'Mul-mut-init\ n \wedge Z ('Muts!j) \}$.

WHILE *True*

INV $\{ 'Mul-mut-init\ n \wedge Z ('Muts!j) \}$.

DO *Mul-Redirect-Edge* *j n* ;;

Mul-Color-Target *j n*

OD

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

Correctness of the proof outline of one mutator

lemma *Mul-Redirect-Edge*: $0 \leq j \wedge j < n \implies$

$\vdash \text{Mul-Redirect-Edge } j\ n$

$\text{pre}(\text{Mul-Color-Target } j\ n)$

$\langle \text{proof} \rangle$

lemma *Mul-Color-Target*: $0 \leq j \wedge j < n \implies$

$\vdash \text{Mul-Color-Target } j\ n$

$\{ 'Mul-mut-init\ n \wedge Z ('Muts!j) \}$.

$\langle \text{proof} \rangle$

lemma *Mul-Mutator*: $0 \leq j \wedge j < n \implies$

$\vdash \text{Mul-Mutator } j\ n.\{False\}$.

$\langle \text{proof} \rangle$

Interference freedom between mutators

lemma *Mul-interfree-Redirect-Edge-Redirect-Edge*:

$\llbracket 0 \leq i; i < n; 0 \leq j; j < n; i \neq j \rrbracket \implies$

interfree-aux (Some (Mul-Redirect-Edge *i n*), {}, Some (Mul-Redirect-Edge *j n*))
 <proof>

lemma *Mul-interfree-Redirect-Edge-Color-Target*:
 $\llbracket 0 \leq i; i < n; 0 \leq j; j < n; i \neq j \rrbracket \implies$
interfree-aux (Some (Mul-Redirect-Edge *i n*), {}, Some (Mul-Color-Target *j n*))
 <proof>

lemma *Mul-interfree-Color-Target-Redirect-Edge*:
 $\llbracket 0 \leq i; i < n; 0 \leq j; j < n; i \neq j \rrbracket \implies$
interfree-aux (Some (Mul-Color-Target *i n*), {}, Some (Mul-Redirect-Edge *j n*))
 <proof>

lemma *Mul-interfree-Color-Target-Color-Target*:
 $\llbracket 0 \leq i; i < n; 0 \leq j; j < n; i \neq j \rrbracket \implies$
interfree-aux (Some (Mul-Color-Target *i n*), {}, Some (Mul-Color-Target *j n*))
 <proof>

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

lemma *Mul-interfree-Mutator-Mutator*: $\llbracket i < n; j < n; i \neq j \rrbracket \implies$
interfree-aux (Some (Mul-Mutator *i n*), {}, Some (Mul-Mutator *j n*))
 <proof>

Modular Parameterized Mutators

lemma *Mul-Parameterized-Mutators*: $0 < n \implies$
 $\llbracket - \cdot \{ 'Mul-mut-init\ n \wedge (\forall i < n. Z\ i\ 'Muts!i) \} \rrbracket$.
 COBEGIN
 SCHEME $[0 \leq j < n]$
Mul-Mutator j n
 $\cdot \{ False \}$.
 COEND
 $\cdot \{ False \}$.
 <proof>

2.3.2 The Collector

constdefs
Queue :: *mul-gar-coll-state* \Rightarrow *nat*
Queue $\equiv \ll length\ (filter\ (\lambda i. \neg Z\ i \wedge 'M!(T\ i) \neq Black)\ 'Muts) \gg$

consts *M-init* :: *nodes*

constdefs
Proper-M-init :: *mul-gar-coll-state* \Rightarrow *bool*
Proper-M-init $\equiv \ll Blacks\ M-init = Roots \wedge length\ M-init = length\ 'M \gg$

$Mul-Prop\text{er} :: mul\text{-}gar\text{-}coll\text{-}state \Rightarrow nat \Rightarrow bool$
 $Mul-Prop\text{er} \equiv \ll \lambda n. Proper\text{-}Roots\ 'M \wedge Proper\text{-}Edges\ ('M, 'E) \wedge 'Proper\text{-}M\text{-}init$
 $\wedge n=length\ 'Muts \gg$

$Safe :: mul\text{-}gar\text{-}coll\text{-}state \Rightarrow bool$
 $Safe \equiv \ll Reach\ 'E \subseteq Blacks\ 'M \gg$

lemmas $mul\text{-}collector\text{-}defs = Proper\text{-}M\text{-}init\text{-}def\ Mul\text{-}Prop\text{er}\text{-}def\ Safe\text{-}def$

Blackening Roots

constdefs

$Mul\text{-}Blacken\text{-}Roots :: nat \Rightarrow mul\text{-}gar\text{-}coll\text{-}state\ ann\text{-}com$
 $Mul\text{-}Blacken\text{-}Roots\ n \equiv$
 $\{ 'Mul\text{-}Prop\text{er}\ n \}.$
 $'ind:=0;;$
 $\{ 'Mul\text{-}Prop\text{er}\ n \wedge 'ind=0 \}.$
 $WHILE\ 'ind < length\ 'M$
 $INV\ \{ 'Mul\text{-}Prop\text{er}\ n \wedge (\forall i < 'ind. i \in Roots \longrightarrow 'M!i=Black) \wedge 'ind \leq length$
 $'M \}.$
 $DO\ \{ 'Mul\text{-}Prop\text{er}\ n \wedge (\forall i < 'ind. i \in Roots \longrightarrow 'M!i=Black) \wedge 'ind < length$
 $'M \}.$
 $IF\ 'ind \in Roots\ THEN$
 $\{ 'Mul\text{-}Prop\text{er}\ n \wedge (\forall i < 'ind. i \in Roots \longrightarrow 'M!i=Black) \wedge 'ind < length\ 'M$
 $\wedge 'ind \in Roots \}.$
 $'M := 'M['ind := Black]\ FI;;$
 $\{ 'Mul\text{-}Prop\text{er}\ n \wedge (\forall i < 'ind+1. i \in Roots \longrightarrow 'M!i=Black) \wedge 'ind < length$
 $'M \}.$
 $'ind := 'ind+1$
 OD

lemma $Mul\text{-}Blacken\text{-}Roots:$

$\vdash Mul\text{-}Blacken\text{-}Roots\ n$
 $\{ 'Mul\text{-}Prop\text{er}\ n \wedge Roots \subseteq Blacks\ 'M \}.$
 $\langle proof \rangle$

Propagating Black

constdefs

$Mul\text{-}PBInv :: mul\text{-}gar\text{-}coll\text{-}state \Rightarrow bool$
 $Mul\text{-}PBInv \equiv \ll 'Safe \vee 'obc \subseteq Blacks\ 'M \vee 'l < 'Queue$
 $\vee (\forall i < 'ind. \neg BtoW('E!i, 'M)) \wedge 'l \leq 'Queue \gg$

 $Mul\text{-}Auxk :: mul\text{-}gar\text{-}coll\text{-}state \Rightarrow bool$
 $Mul\text{-}Auxk \equiv \ll 'l < 'Queue \vee 'M! 'k \neq Black \vee \neg BtoW('E! 'ind, 'M) \vee 'obc \subseteq Blacks$
 $'M \gg$

constdefs

$Mul\text{-}Propagate\text{-}Black :: nat \Rightarrow mul\text{-}gar\text{-}coll\text{-}state\ ann\text{-}com$
 $Mul\text{-}Propagate\text{-}Black\ n \equiv$

```

.{ 'Mul-Prop $er$   $n \wedge \text{Roots} \subseteq \text{Blacks } 'M \wedge 'obc \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$ 
 $\wedge ( 'Safe \vee 'l \leq 'Queue \vee 'obc \subseteq \text{Blacks } 'M )$  }.
'ind:=0;;
.{ 'Mul-Prop $er$   $n \wedge \text{Roots} \subseteq \text{Blacks } 'M$ 
 $\wedge 'obc \subseteq \text{Blacks } 'M \wedge \text{Blacks } 'M \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$ 
 $\wedge ( 'Safe \vee 'l \leq 'Queue \vee 'obc \subseteq \text{Blacks } 'M ) \wedge 'ind=0$  }.
WHILE 'ind<length 'E
INV .{ 'Mul-Prop $er$   $n \wedge \text{Roots} \subseteq \text{Blacks } 'M$ 
 $\wedge 'obc \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$ 
 $\wedge 'Mul-PBInv \wedge 'ind \leq \text{length } 'E$  }.
DO .{ 'Mul-Prop $er$   $n \wedge \text{Roots} \subseteq \text{Blacks } 'M$ 
 $\wedge 'obc \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$ 
 $\wedge 'Mul-PBInv \wedge 'ind < \text{length } 'E$  }.
IF 'M!(fst ('E!'ind))=Black THEN
.{ 'Mul-Prop $er$   $n \wedge \text{Roots} \subseteq \text{Blacks } 'M$ 
 $\wedge 'obc \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$ 
 $\wedge 'Mul-PBInv \wedge ( 'M!fst('E!'ind))=Black \wedge 'ind < \text{length } 'E$  }.
'k:=snd('E!'ind);;
.{ 'Mul-Prop $er$   $n \wedge \text{Roots} \subseteq \text{Blacks } 'M$ 
 $\wedge 'obc \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$ 
 $\wedge ( 'Safe \vee 'obc \subseteq \text{Blacks } 'M \vee 'l < 'Queue \vee (\forall i < 'ind. \neg BtoW('E!i, 'M))$ 
 $\wedge 'l \leq 'Queue \wedge 'Mul-Auxk ) \wedge 'k < \text{length } 'M \wedge 'M!fst('E!'ind)=Black$ 
 $\wedge 'ind < \text{length } 'E$  }.
<'M:='M['k:=Black],, 'ind:='ind+1>
ELSE .{ 'Mul-Prop $er$   $n \wedge \text{Roots} \subseteq \text{Blacks } 'M$ 
 $\wedge 'obc \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$ 
 $\wedge 'Mul-PBInv \wedge 'ind < \text{length } 'E$  }.
<IF 'M!(fst ('E!'ind)) $\neq$ Black THEN 'ind:='ind+1 FI> FI
OD

```

lemma *Mul-Propagate-Black*:

```

 $\vdash \text{Mul-Propagate-Black } n$ 
. $\{ 'Mul-Prop $er$   $n \wedge \text{Roots} \subseteq \text{Blacks } 'M \wedge 'obc \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$ 
 $\wedge ( 'Safe \vee 'obc \subseteq \text{Blacks } 'M \vee 'l < 'Queue \wedge ( 'l \leq 'Queue \vee 'obc \subseteq \text{Blacks } 'M ) )$  }.$ 
```

<proof>

Counting Black Nodes

constdefs

```

Mul-CountInv :: mul-gar-coll-state  $\Rightarrow$  nat  $\Rightarrow$  bool
Mul-CountInv  $\equiv \ll \lambda ind. \{ i. i < ind \wedge 'Ma!i=Black \} \subseteq 'bc \gg$ 

```

```

Mul-Count :: nat  $\Rightarrow$  mul-gar-coll-state ann-com
Mul-Count n  $\equiv$ 
. $\{ 'Mul-Prop $er$   $n \wedge \text{Roots} \subseteq \text{Blacks } 'M$ 
 $\wedge 'obc \subseteq \text{Blacks } 'Ma \wedge \text{Blacks } 'Ma \subseteq \text{Blacks } 'M \wedge 'bc \subseteq \text{Blacks } 'M$ 
 $\wedge \text{length } 'Ma = \text{length } 'M$ 
 $\wedge ( 'Safe \vee 'obc \subseteq \text{Blacks } 'Ma \vee 'l < 'q \wedge ( 'q \leq 'Queue \vee 'obc \subseteq \text{Blacks } 'M ) )$  }$ 
```

$\wedge 'q < n+1 \wedge 'bc = \{\}$.
 $'ind := 0;;$
 $\{ 'Mul\text{-}Proper\ n \wedge Roots \subseteq Blacks\ 'M$
 $\wedge 'obc \subseteq Blacks\ 'Ma \wedge Blacks\ 'Ma \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge length\ 'Ma = length\ 'M$
 $\wedge ('Safe \vee 'obc \subseteq Blacks\ 'Ma \vee 'l < 'q \wedge ('q \leq 'Queue \vee 'obc \subseteq Blacks\ 'M))$
 $\wedge 'q < n+1 \wedge 'bc = \{\} \wedge 'ind = 0\}.$
WHILE $'ind < length\ 'M$
 $\text{INV } \{ 'Mul\text{-}Proper\ n \wedge Roots \subseteq Blacks\ 'M$
 $\wedge 'obc \subseteq Blacks\ 'Ma \wedge Blacks\ 'Ma \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge length\ 'Ma = length\ 'M \wedge 'Mul\text{-}CountInv\ 'ind$
 $\wedge ('Safe \vee 'obc \subseteq Blacks\ 'Ma \vee 'l < 'q \wedge ('q \leq 'Queue \vee 'obc \subseteq Blacks\ 'M))$
 $'M))$
 $\wedge 'q < n+1 \wedge 'ind \leq length\ 'M\}.$
DO $\{ 'Mul\text{-}Proper\ n \wedge Roots \subseteq Blacks\ 'M$
 $\wedge 'obc \subseteq Blacks\ 'Ma \wedge Blacks\ 'Ma \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge length\ 'Ma = length\ 'M \wedge 'Mul\text{-}CountInv\ 'ind$
 $\wedge ('Safe \vee 'obc \subseteq Blacks\ 'Ma \vee 'l < 'q \wedge ('q \leq 'Queue \vee 'obc \subseteq Blacks\ 'M))$
 $\wedge 'q < n+1 \wedge 'ind < length\ 'M\}.$
IF $'M! 'ind = Black$
THEN $\{ 'Mul\text{-}Proper\ n \wedge Roots \subseteq Blacks\ 'M$
 $\wedge 'obc \subseteq Blacks\ 'Ma \wedge Blacks\ 'Ma \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge length\ 'Ma = length\ 'M \wedge 'Mul\text{-}CountInv\ 'ind$
 $\wedge ('Safe \vee 'obc \subseteq Blacks\ 'Ma \vee 'l < 'q \wedge ('q \leq 'Queue \vee 'obc \subseteq Blacks\ 'M))$
 $'M))$
 $\wedge 'q < n+1 \wedge 'ind < length\ 'M \wedge 'M! 'ind = Black\}.$
 $'bc := insert\ 'ind\ 'bc$
FI;;
 $\{ 'Mul\text{-}Proper\ n \wedge Roots \subseteq Blacks\ 'M$
 $\wedge 'obc \subseteq Blacks\ 'Ma \wedge Blacks\ 'Ma \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge length\ 'Ma = length\ 'M \wedge 'Mul\text{-}CountInv\ ('ind+1)$
 $\wedge ('Safe \vee 'obc \subseteq Blacks\ 'Ma \vee 'l < 'q \wedge ('q \leq 'Queue \vee 'obc \subseteq Blacks\ 'M))$
 $\wedge 'q < n+1 \wedge 'ind < length\ 'M\}.$
 $'ind := 'ind+1$
OD

lemma *Mul-Count*:

$\vdash Mul\text{-}Count\ n$

$\{ 'Mul\text{-}Proper\ n \wedge Roots \subseteq Blacks\ 'M$
 $\wedge 'obc \subseteq Blacks\ 'Ma \wedge Blacks\ 'Ma \subseteq Blacks\ 'M \wedge 'bc \subseteq Blacks\ 'M$
 $\wedge length\ 'Ma = length\ 'M \wedge Blacks\ 'Ma \subseteq 'bc$
 $\wedge ('Safe \vee 'obc \subseteq Blacks\ 'Ma \vee 'l < 'q \wedge ('q \leq 'Queue \vee 'obc \subseteq Blacks\ 'M))$
 $\wedge 'q < n+1\}.$

<proof>

Appending garbage nodes to the free list

consts *Append-to-free* :: $nat \times edges \Rightarrow edges$

axioms

$Append\text{-}to\text{-}free0: length (Append\text{-}to\text{-}free (i, e)) = length e$
 $Append\text{-}to\text{-}free1: Proper\text{-}Edges (m, e)$
 $\implies Proper\text{-}Edges (m, Append\text{-}to\text{-}free(i, e))$
 $Append\text{-}to\text{-}free2: i \notin Reach e$
 $\implies n \in Reach (Append\text{-}to\text{-}free(i, e)) = (n = i \vee n \in Reach e)$

constdefs

$Mul\text{-}AppendInv :: mul\text{-}gar\text{-}coll\text{-}state \Rightarrow nat \Rightarrow bool$
 $Mul\text{-}AppendInv \equiv \ll \lambda ind. (\forall i. ind \leq i \longrightarrow i < length \text{'M} \longrightarrow i \in Reach \text{'E} \longrightarrow \text{'M}!i = Black) \gg$

 $Mul\text{-}Append :: nat \Rightarrow mul\text{-}gar\text{-}coll\text{-}state \text{ ann-com}$
 $Mul\text{-}Append n \equiv$
 $\{ \text{'Mul-}Proper\ n \wedge Roots \subseteq Blacks \text{'M} \wedge \text{'Safe} \}.$
 $\text{'ind} := 0;;$
 $\{ \text{'Mul-}Proper\ n \wedge Roots \subseteq Blacks \text{'M} \wedge \text{'Safe} \wedge \text{'ind} = 0 \}.$
 $WHILE \text{'ind} < length \text{'M}$
 $INV \{ \text{'Mul-}Proper\ n \wedge \text{'Mul-}AppendInv \text{'ind} \wedge \text{'ind} \leq length \text{'M} \}.$
 $DO \{ \text{'Mul-}Proper\ n \wedge \text{'Mul-}AppendInv \text{'ind} \wedge \text{'ind} < length \text{'M} \}.$
 $IF \text{'M}! \text{'ind} = Black THEN$
 $\{ \text{'Mul-}Proper\ n \wedge \text{'Mul-}AppendInv \text{'ind} \wedge \text{'ind} < length \text{'M} \wedge \text{'M}! \text{'ind} = Black \}.$

 $\text{'M} := \text{'M}[\text{'ind} := White]$
 $ELSE$
 $\{ \text{'Mul-}Proper\ n \wedge \text{'Mul-}AppendInv \text{'ind} \wedge \text{'ind} < length \text{'M} \wedge \text{'ind} \notin Reach$
 $\text{'E} \}.$
 $\text{'E} := Append\text{-}to\text{-}free(\text{'ind}, \text{'E})$
 $FI;;$
 $\{ \text{'Mul-}Proper\ n \wedge \text{'Mul-}AppendInv (\text{'ind} + 1) \wedge \text{'ind} < length \text{'M} \}.$
 $\text{'ind} := \text{'ind} + 1$
 OD

lemma *Mul-Append*:

$\vdash Mul\text{-}Append\ n$
 $\{ \text{'Mul-}Proper\ n \}.$
 $\langle proof \rangle$

Collector

constdefs

$Mul\text{-}Collector :: nat \Rightarrow mul\text{-}gar\text{-}coll\text{-}state \text{ ann-com}$
 $Mul\text{-}Collector n \equiv$
 $\{ \text{'Mul-}Proper\ n \}.$
 $WHILE True INV \{ \text{'Mul-}Proper\ n \}.$
 DO
 $Mul\text{-}Blacken\text{-}Roots\ n ;;$
 $\{ \text{'Mul-}Proper\ n \wedge Roots \subseteq Blacks \text{'M} \}.$
 $\text{'obc} := \{ \};;$


```

. $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M \wedge 'obc = \{\} \}$ .
   $'bc := Roots;$ 
. $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M \wedge 'obc = \{\} \wedge 'bc = Roots \}$ .
   $'l := 0;$ 
. $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M \wedge 'obc = \{\} \wedge 'bc = Roots \wedge 'l = 0 \}$ .
  WHILE  $'l < n + 1$ 
    INV  $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M \wedge 'bc \subseteq Blacks \textit{' } M \wedge$ 
       $( 'Safe \vee ('l \leq 'Queue \vee 'bc \subseteq Blacks \textit{' } M) \wedge 'l < n + 1) \}$ .
  DO  $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M \wedge 'bc \subseteq Blacks \textit{' } M$ 
     $\wedge ( 'Safe \vee 'l \leq 'Queue \vee 'bc \subseteq Blacks \textit{' } M) \}$ .
     $'obc := 'bc;$ 
    Mul-Propagate-Black  $n;$ 
     $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M$ 
       $\wedge 'obc \subseteq Blacks \textit{' } M \wedge 'bc \subseteq Blacks \textit{' } M$ 
       $\wedge ( 'Safe \vee 'obc \subseteq Blacks \textit{' } M \vee 'l < 'Queue$ 
       $\wedge ('l \leq 'Queue \vee 'obc \subseteq Blacks \textit{' } M) ) \}$ .
     $'bc := \{\};$ 
     $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M$ 
       $\wedge 'obc \subseteq Blacks \textit{' } M \wedge 'bc \subseteq Blacks \textit{' } M$ 
       $\wedge ( 'Safe \vee 'obc \subseteq Blacks \textit{' } M \vee 'l < 'Queue$ 
       $\wedge ('l \leq 'Queue \vee 'obc \subseteq Blacks \textit{' } M) ) \wedge 'bc = \{\} \}$ .
     $\langle 'Ma := 'M, 'q := 'Queue \rangle;$ 
    Mul-Count  $n;$ 
     $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M$ 
       $\wedge 'obc \subseteq Blacks \textit{' } Ma \wedge Blacks \textit{' } Ma \subseteq Blacks \textit{' } M \wedge 'bc \subseteq Blacks \textit{' } M$ 
       $\wedge length \textit{' } Ma = length \textit{' } M \wedge Blacks \textit{' } Ma \subseteq 'bc$ 
       $\wedge ( 'Safe \vee 'obc \subseteq Blacks \textit{' } Ma \vee 'l < 'q \wedge ('q \leq 'Queue \vee 'obc \subseteq Blacks \textit{' } M) )$ 
       $\wedge 'q < n + 1 \}$ .
    IF  $'obc = 'bc$  THEN
       $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M$ 
         $\wedge 'obc \subseteq Blacks \textit{' } Ma \wedge Blacks \textit{' } Ma \subseteq Blacks \textit{' } M \wedge 'bc \subseteq Blacks \textit{' } M$ 
         $\wedge length \textit{' } Ma = length \textit{' } M \wedge Blacks \textit{' } Ma \subseteq 'bc$ 
         $\wedge ( 'Safe \vee 'obc \subseteq Blacks \textit{' } Ma \vee 'l < 'q \wedge ('q \leq 'Queue \vee 'obc \subseteq Blacks \textit{' } M) )$ 
         $\wedge 'q < n + 1 \wedge 'obc = 'bc \}$ .
       $'l := 'l + 1$ 
    ELSE  $\{ 'Mul-Prop\textit{er } n \wedge Roots \subseteq Blacks \textit{' } M$ 
       $\wedge 'obc \subseteq Blacks \textit{' } Ma \wedge Blacks \textit{' } Ma \subseteq Blacks \textit{' } M \wedge 'bc \subseteq Blacks \textit{' } M$ 
       $\wedge length \textit{' } Ma = length \textit{' } M \wedge Blacks \textit{' } Ma \subseteq 'bc$ 
       $\wedge ( 'Safe \vee 'obc \subseteq Blacks \textit{' } Ma \vee 'l < 'q \wedge ('q \leq 'Queue \vee 'obc \subseteq Blacks$ 
       $\textit{' } M) )$ 
       $\wedge 'q < n + 1 \wedge 'obc \neq 'bc \}$ .
       $'l := 0$  FI
    OD;;
  Mul-Append  $n$ 
  OD

```

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

lemma *Mul-Collector*:

$\vdash \text{Mul-Collector } n$
 $\cdot \{False\}.$
 $\langle \text{proof} \rangle$

2.3.3 Interference Freedom

lemma *le-length-filter-update*[*rule-format*]:

$\forall i. (\neg P \text{ (list!}i) \vee P \text{ } j) \wedge i < \text{length list}$
 $\longrightarrow \text{length}(\text{filter } P \text{ list}) \leq \text{length}(\text{filter } P \text{ (list[i:=j])})$
 $\langle \text{proof} \rangle$

lemma *less-length-filter-update* [*rule-format*]:

$\forall i. P \text{ } j \wedge \neg(P \text{ (list!}i)) \wedge i < \text{length list}$
 $\longrightarrow \text{length}(\text{filter } P \text{ list}) < \text{length}(\text{filter } P \text{ (list[i:=j])})$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Blacken-Roots-Redirect-Edge*: $\llbracket 0 \leq j; j < n \rrbracket \Longrightarrow$

$\text{interfree-aux } (\text{Some}(\text{Mul-Blacken-Roots } n), \{\}, \text{Some}(\text{Mul-Redirect-Edge } j \text{ } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Redirect-Edge-Blacken-Roots*: $\llbracket 0 \leq j; j < n \rrbracket \Longrightarrow$

$\text{interfree-aux } (\text{Some}(\text{Mul-Redirect-Edge } j \text{ } n), \{\}, \text{Some } (\text{Mul-Blacken-Roots } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Blacken-Roots-Color-Target*: $\llbracket 0 \leq j; j < n \rrbracket \Longrightarrow$

$\text{interfree-aux } (\text{Some}(\text{Mul-Blacken-Roots } n), \{\}, \text{Some } (\text{Mul-Color-Target } j \text{ } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Color-Target-Blacken-Roots*: $\llbracket 0 \leq j; j < n \rrbracket \Longrightarrow$

$\text{interfree-aux } (\text{Some}(\text{Mul-Color-Target } j \text{ } n), \{\}, \text{Some } (\text{Mul-Blacken-Roots } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Propagate-Black-Redirect-Edge*: $\llbracket 0 \leq j; j < n \rrbracket \Longrightarrow$

$\text{interfree-aux } (\text{Some}(\text{Mul-Propagate-Black } n), \{\}, \text{Some } (\text{Mul-Redirect-Edge } j \text{ } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Redirect-Edge-Propagate-Black*: $\llbracket 0 \leq j; j < n \rrbracket \Longrightarrow$

$\text{interfree-aux } (\text{Some}(\text{Mul-Redirect-Edge } j \text{ } n), \{\}, \text{Some } (\text{Mul-Propagate-Black } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Propagate-Black-Color-Target*: $\llbracket 0 \leq j; j < n \rrbracket \Longrightarrow$

$\text{interfree-aux } (\text{Some}(\text{Mul-Propagate-Black } n), \{\}, \text{Some } (\text{Mul-Color-Target } j \text{ } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Color-Target-Propagate-Black*: $\llbracket 0 \leq j; j < n \rrbracket \Longrightarrow$

$\text{interfree-aux } (\text{Some}(\text{Mul-Color-Target } j \text{ } n), \{\}, \text{Some}(\text{Mul-Propagate-Black } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Count-Redirect-Edge*: $\llbracket 0 \leq j; j < n \rrbracket \implies$
 $\text{interfree-aux } (\text{Some}(\text{Mul-Count } n), \{\}, \text{Some}(\text{Mul-Redirect-Edge } j \ n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Redirect-Edge-Count*: $\llbracket 0 \leq j; j < n \rrbracket \implies$
 $\text{interfree-aux } (\text{Some}(\text{Mul-Redirect-Edge } j \ n), \{\}, \text{Some}(\text{Mul-Count } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Count-Color-Target*: $\llbracket 0 \leq j; j < n \rrbracket \implies$
 $\text{interfree-aux } (\text{Some}(\text{Mul-Count } n), \{\}, \text{Some}(\text{Mul-Color-Target } j \ n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Color-Target-Count*: $\llbracket 0 \leq j; j < n \rrbracket \implies$
 $\text{interfree-aux } (\text{Some}(\text{Mul-Color-Target } j \ n), \{\}, \text{Some}(\text{Mul-Count } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Append-Redirect-Edge*: $\llbracket 0 \leq j; j < n \rrbracket \implies$
 $\text{interfree-aux } (\text{Some}(\text{Mul-Append } n), \{\}, \text{Some}(\text{Mul-Redirect-Edge } j \ n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Redirect-Edge-Append*: $\llbracket 0 \leq j; j < n \rrbracket \implies$
 $\text{interfree-aux } (\text{Some}(\text{Mul-Redirect-Edge } j \ n), \{\}, \text{Some}(\text{Mul-Append } n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Append-Color-Target*: $\llbracket 0 \leq j; j < n \rrbracket \implies$
 $\text{interfree-aux } (\text{Some}(\text{Mul-Append } n), \{\}, \text{Some}(\text{Mul-Color-Target } j \ n))$
 $\langle \text{proof} \rangle$

lemma *Mul-interfree-Color-Target-Append*: $\llbracket 0 \leq j; j < n \rrbracket \implies$
 $\text{interfree-aux } (\text{Some}(\text{Mul-Color-Target } j \ n), \{\}, \text{Some}(\text{Mul-Append } n))$
 $\langle \text{proof} \rangle$

Interference freedom Collector-Mutator

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

Mul-interfree-Propagate-Black-Redirect-Edge *Mul-interfree-Propagate-Black-Color-Target*

Mul-interfree-Count-Redirect-Edge *Mul-interfree-Count-Color-Target*

Mul-interfree-Append-Redirect-Edge *Mul-interfree-Append-Color-Target*

Mul-interfree-Redirect-Edge-Blacken-Roots *Mul-interfree-Color-Target-Blacken-Roots*

Mul-interfree-Redirect-Edge-Propagate-Black *Mul-interfree-Color-Target-Propagate-Black*

Mul-interfree-Redirect-Edge-Count *Mul-interfree-Color-Target-Count*

Mul-interfree-Redirect-Edge-Append *Mul-interfree-Color-Target-Append*

lemma *Mul-interfree-Collector-Mutator*: $j < n \implies$
 $\text{interfree-aux } (\text{Some } (\text{Mul-Collector } n), \{\}, \text{Some } (\text{Mul-Mutator } j \ n))$
 $\langle \text{proof} \rangle$

Interference freedom Mutator-Collector

lemma *Mul-interfree-Mutator-Collector*: $j < n \implies$
 $\text{interfree-aux } (\text{Some } (\text{Mul-Mutator } j \ n), \{\}, \text{Some } (\text{Mul-Collector } n))$
 $\langle \text{proof} \rangle$

The Multi-Mutator Garbage Collection Algorithm

The total number of verification conditions is 328

lemma *Mul-Gar-Coll*:
 $\| - .\{ ' \text{Mul-Prop} \text{er } n \wedge ' \text{Mul-mut-init } n \wedge (\forall i < n. Z (' \text{Muts!} i)) \}.$
 COBEGIN
 $\text{Mul-Collector } n$
 $.\{ \text{False} \}.$
 $\|$
 $\text{SCHEME } [0 \leq j < n]$
 $\text{Mul-Mutator } j \ n$
 $.\{ \text{False} \}.$
 COEND
 $.\{ \text{False} \}.$
 $\langle \text{proof} \rangle$
end

Chapter 3

The Rely-Guarantee Method

3.1 Abstract Syntax

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

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

types

'a bexp = *'a set*

datatype *'a com* =

Basic 'a \Rightarrow *'a*

| *Seq 'a com 'a com*

| *Cond 'a bexp 'a com 'a com*

| *While 'a bexp 'a com*

| *Await 'a bexp 'a com*

types *'a par-com* = ((*'a com*) *option*) *list*

end

3.2 Operational Semantics

theory *RG-Tran*

imports *RG-Com*

begin

3.2.1 Semantics of Component Programs

Environment transitions

types *'a conf* = ((*'a com*) *option*) \times *'a*

inductive-set

etran :: (*'a conf* \times *'a conf*) *set*

and $etran' :: 'a\ conf \Rightarrow 'a\ conf \Rightarrow bool \quad (- -e\rightarrow - [81,81] 80)$

where

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

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

Component transitions

inductive-set

$ctran :: ('a\ conf \times 'a\ conf)\ set$
and $ctran' :: 'a\ conf \Rightarrow 'a\ conf \Rightarrow bool \quad (- -c\rightarrow - [81,81] 80)$
and $ctrans :: 'a\ conf \Rightarrow 'a\ conf \Rightarrow bool \quad (- -c*\rightarrow - [81,81] 80)$

where

$P -c\rightarrow Q \equiv (P, Q) \in ctran$
 $| P -c*\rightarrow Q \equiv (P, Q) \in ctran^*$

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

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

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

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

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

$| WhileF: s \notin b \Longrightarrow (Some(While\ b\ P), s) -c\rightarrow (None, s)$

$| WhileT: s \in b \Longrightarrow (Some(While\ b\ P), s) -c\rightarrow (Some(Seq\ P\ (While\ b\ P)), s)$

$| Await: \llbracket s \in b; (Some\ P, s) -c*\rightarrow (None, t) \rrbracket \Longrightarrow (Some(Await\ b\ P), s) -c\rightarrow (None, t)$

monos $rtrancl\text{-}mono$

3.2.2 Semantics of Parallel Programs

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

inductive-set

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

where

$P -pe\rightarrow Q \equiv (P, Q) \in par\text{-}etran$
 $| ParEnv: (Ps, s) -pe\rightarrow (Ps, t)$

inductive-set

$par\text{-}ctran :: ('a\ par\text{-}conf \times 'a\ par\text{-}conf)\ set$

and $\text{par-ctran}' :: ['a \text{ par-conf}, 'a \text{ par-conf}] \Rightarrow \text{bool} \ (- \text{-pc} \rightarrow - [81,81] \ 80)$
where
 $P \text{-pc} \rightarrow Q \equiv (P, Q) \in \text{par-ctran}$
 $| \text{ParComp}: \llbracket i < \text{length } Ps; (Ps[i], s) \text{-c} \rightarrow (r, t) \rrbracket \Longrightarrow (Ps, s) \text{-pc} \rightarrow (Ps[i:=r], t)$
lemma $\text{par-ctranE}: c \text{-pc} \rightarrow c' \Longrightarrow$
 $(\bigwedge i \text{ Ps } s \ r \ t. c = (Ps, s) \Longrightarrow c' = (Ps[i := r], t) \Longrightarrow i < \text{length } Ps \Longrightarrow$
 $(Ps[i], s) \text{-c} \rightarrow (r, t) \Longrightarrow P) \Longrightarrow P$
 $\langle \text{proof} \rangle$

3.2.3 Computations

Sequential computations

types $'a \text{ confs} = ('a \text{ conf}) \text{ list}$

inductive-set $\text{cptn} :: ('a \text{ confs}) \text{ set}$

where

$\text{CptnOne}: [(P, s)] \in \text{cptn}$
 $| \text{CptnEnv}: (P, t) \# xs \in \text{cptn} \Longrightarrow (P, s) \# (P, t) \# xs \in \text{cptn}$
 $| \text{CptnComp}: \llbracket (P, s) \text{-c} \rightarrow (Q, t); (Q, t) \# xs \in \text{cptn} \rrbracket \Longrightarrow (P, s) \# (Q, t) \# xs \in \text{cptn}$

constdefs

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

Parallel computations

types $'a \text{ par-confs} = ('a \text{ par-conf}) \text{ list}$

inductive-set $\text{par-cptn} :: ('a \text{ par-confs}) \text{ set}$

where

$\text{ParCptnOne}: [(P, s)] \in \text{par-cptn}$
 $| \text{ParCptnEnv}: (P, t) \# xs \in \text{par-cptn} \Longrightarrow (P, s) \# (P, t) \# xs \in \text{par-cptn}$
 $| \text{ParCptnComp}: \llbracket (P, s) \text{-pc} \rightarrow (Q, t); (Q, t) \# xs \in \text{par-cptn} \rrbracket \Longrightarrow (P, s) \# (Q, t) \# xs \in \text{par-cptn}$

constdefs

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

3.2.4 Modular Definition of Computation

constdefs

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

inductive-set $\text{cptn-mod} :: ('a \text{ confs}) \text{ set}$

where

$\text{CptnModOne}: [(P, s)] \in \text{cptn-mod}$

$| \text{CptnModEnv}: (P, t) \# xs \in \text{cptn-mod} \implies (P, s) \# (P, t) \# xs \in \text{cptn-mod}$
 $| \text{CptnModNone}: \llbracket (\text{Some } P, s) -c \rightarrow (\text{None}, t); (\text{None}, t) \# xs \in \text{cptn-mod} \rrbracket \implies$
 $(\text{Some } P, s) \# (\text{None}, t) \# xs \in \text{cptn-mod}$
 $| \text{CptnModCondT}: \llbracket (\text{Some } P0, s) \# ys \in \text{cptn-mod}; s \in b \rrbracket \implies (\text{Some}(\text{Cond } b \ P0$
 $P1), s) \# (\text{Some } P0, s) \# ys \in \text{cptn-mod}$
 $| \text{CptnModCondF}: \llbracket (\text{Some } P1, s) \# ys \in \text{cptn-mod}; s \notin b \rrbracket \implies (\text{Some}(\text{Cond } b \ P0$
 $P1), s) \# (\text{Some } P1, s) \# ys \in \text{cptn-mod}$
 $| \text{CptnModSeq1}: \llbracket (\text{Some } P0, s) \# xs \in \text{cptn-mod}; zs = \text{map } (\text{lift } P1) \ xs \rrbracket$
 $\implies (\text{Some}(\text{Seq } P0 \ P1), s) \# zs \in \text{cptn-mod}$
 $| \text{CptnModSeq2}:$
 $\llbracket (\text{Some } P0, s) \# xs \in \text{cptn-mod}; \text{fst}(\text{last } ((\text{Some } P0, s) \# xs)) = \text{None};$
 $(\text{Some } P1, \text{snd}(\text{last } ((\text{Some } P0, s) \# xs))) \# ys \in \text{cptn-mod};$
 $zs = (\text{map } (\text{lift } P1) \ xs) @ ys \rrbracket \implies (\text{Some}(\text{Seq } P0 \ P1), s) \# zs \in \text{cptn-mod}$
 $| \text{CptnModWhile1}:$
 $\llbracket (\text{Some } P, s) \# xs \in \text{cptn-mod}; s \in b; zs = \text{map } (\text{lift } (\text{While } b \ P)) \ xs \rrbracket$
 $\implies (\text{Some}(\text{While } b \ P), s) \# (\text{Some}(\text{Seq } P \ (\text{While } b \ P)), s) \# zs \in \text{cptn-mod}$
 $| \text{CptnModWhile2}:$
 $\llbracket (\text{Some } P, s) \# xs \in \text{cptn-mod}; \text{fst}(\text{last } ((\text{Some } P, s) \# xs)) = \text{None}; s \in b;$
 $zs = (\text{map } (\text{lift } (\text{While } b \ P)) \ xs) @ ys;$
 $(\text{Some}(\text{While } b \ P), \text{snd}(\text{last } ((\text{Some } P, s) \# xs))) \# ys \in \text{cptn-mod} \rrbracket$
 $\implies (\text{Some}(\text{While } b \ P), s) \# (\text{Some}(\text{Seq } P \ (\text{While } b \ P)), s) \# zs \in \text{cptn-mod}$

3.2.5 Equivalence of Both Definitions.

lemma *last-length*: $((a \# xs)!(\text{length } xs)) = \text{last } (a \# xs)$
 $\langle \text{proof} \rangle$

lemma *div-seq [rule-format]*: $\text{list} \in \text{cptn-mod} \implies$
 $(\forall s \ P \ Q \ zs. \text{list} = (\text{Some } (\text{Seq } P \ Q), s) \# zs \longrightarrow$
 $(\exists xs. (\text{Some } P, s) \# xs \in \text{cptn-mod} \wedge (zs = (\text{map } (\text{lift } Q) \ xs) \vee$
 $(\text{fst}(((\text{Some } P, s) \# xs)!\text{length } xs) = \text{None} \wedge$
 $(\exists ys. (\text{Some } Q, \text{snd}(((\text{Some } P, s) \# xs)!\text{length } xs))) \# ys \in \text{cptn-mod}$
 $\wedge zs = (\text{map } (\text{lift } (Q)) \ xs) @ ys))))$
 $\langle \text{proof} \rangle$

lemma *cptn-onlyif-cptn-mod-aux [rule-format]*:
 $\forall s \ Q \ t \ xs. ((\text{Some } a, s), Q, t) \in \text{ctran} \longrightarrow (Q, t) \# xs \in \text{cptn-mod}$
 $\longrightarrow (\text{Some } a, s) \# (Q, t) \# xs \in \text{cptn-mod}$
 $\langle \text{proof} \rangle$

lemma *cptn-onlyif-cptn-mod [rule-format]*: $c \in \text{cptn} \implies c \in \text{cptn-mod}$
 $\langle \text{proof} \rangle$

lemma *lift-is-cptn*: $c \in \text{cptn} \implies \text{map } (\text{lift } P) \ c \in \text{cptn}$
 $\langle \text{proof} \rangle$

lemma *cptn-append-is-cptn [rule-format]*:
 $\forall b \ a. b \# c1 \in \text{cptn} \longrightarrow a \# c2 \in \text{cptn} \longrightarrow (b \# c1)!\text{length } c1 = a \longrightarrow b \# c1 @ c2 \in \text{cptn}$

$\langle \text{proof} \rangle$

lemma *last-lift*: $\llbracket xs \neq []; fst(xs!(length\ xs - (Suc\ 0))) = None \rrbracket$
 $\implies fst((map\ (lift\ P)\ xs)!(length\ (map\ (lift\ P)\ xs) - (Suc\ 0))) = (Some\ P)$
 $\langle \text{proof} \rangle$

lemma *last-fst* [rule-format]: $P((a \# x) ! length\ x) \longrightarrow \neg P\ a \longrightarrow P\ (x!(length\ x - (Suc\ 0)))$
 $\langle \text{proof} \rangle$

lemma *last-fst-esp*:
 $fst(((Some\ a, s) \# xs)!(length\ xs)) = None \implies fst(xs!(length\ xs - (Suc\ 0))) = None$
 $\langle \text{proof} \rangle$

lemma *last-snd*: $xs \neq [] \implies$
 $snd(((map\ (lift\ P)\ xs)!(length\ (map\ (lift\ P)\ xs) - (Suc\ 0)))) = snd(xs!(length\ xs - (Suc\ 0)))$
 $\langle \text{proof} \rangle$

lemma *Cons-lift*: $(Some\ (Seq\ P\ Q), s) \# (map\ (lift\ Q)\ xs) = map\ (lift\ Q)\ ((Some\ P, s) \# xs)$
 $\langle \text{proof} \rangle$

lemma *Cons-lift-append*:
 $(Some\ (Seq\ P\ Q), s) \# (map\ (lift\ Q)\ xs) @ ys = map\ (lift\ Q)\ ((Some\ P, s) \# xs) @ ys$
 $\langle \text{proof} \rangle$

lemma *lift-nth*: $i < length\ xs \implies map\ (lift\ Q)\ xs ! i = lift\ Q\ (xs ! i)$
 $\langle \text{proof} \rangle$

lemma *snd-lift*: $i < length\ xs \implies snd(lift\ Q\ (xs ! i)) = snd\ (xs ! i)$
 $\langle \text{proof} \rangle$

lemma *cptn-if-cptn-mod*: $c \in \text{cptn-mod} \implies c \in \text{cptn}$
 $\langle \text{proof} \rangle$

theorem *cptn-iff-cptn-mod*: $(c \in \text{cptn}) = (c \in \text{cptn-mod})$
 $\langle \text{proof} \rangle$

3.3 Validity of Correctness Formulas

3.3.1 Validity for Component Programs.

types $'a\ \text{rgformula} = 'a\ \text{com} \times 'a\ \text{set} \times ('a \times 'a)\ \text{set} \times ('a \times 'a)\ \text{set} \times 'a\ \text{set}$

consts

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

$$c!i - e \rightarrow c!(\text{Suc } i) \longrightarrow (\text{snd}(c!i), \text{snd}(c!\text{Suc } i)) \in \text{rely}\}}\}$$

$$\begin{aligned} \text{comm} &:: ('a \times 'a) \text{ set} \times 'a \text{ set} \Rightarrow ('a \text{ confs}) \text{ set} \\ \text{comm} &\equiv \lambda(\text{guar}, \text{post}). \{c. (\forall i. \text{Suc } i < \text{length } c \longrightarrow \\ &\quad c!i - c \rightarrow c!(\text{Suc } i) \longrightarrow (\text{snd}(c!i), \text{snd}(c!\text{Suc } i)) \in \text{guar}) \wedge \\ &\quad (\text{fst}(\text{last } c) = \text{None} \longrightarrow \text{snd}(\text{last } c) \in \text{post})\} \end{aligned}$$

$$\text{com-validity} :: 'a \text{ com} \Rightarrow 'a \text{ set} \Rightarrow ('a \times 'a) \text{ set} \Rightarrow ('a \times 'a) \text{ set} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$$

$$\begin{aligned} &(\models - \text{sat } [-, -, -, -] [60, 0, 0, 0, 0] \ 45) \\ \models P \text{ sat } [\text{pre}, \text{rely}, \text{guar}, \text{post}] &\equiv \\ \forall s. \text{cp } (\text{Some } P) \ s \cap \text{assum}(\text{pre}, \text{rely}) &\subseteq \text{comm}(\text{guar}, \text{post}) \end{aligned}$$

3.3.2 Validity for Parallel Programs.

constdefs

$$\begin{aligned} \text{All-None} &:: (('a \text{ com}) \text{ option}) \text{ list} \Rightarrow \text{bool} \\ \text{All-None } xs &\equiv \forall c \in \text{set } xs. c = \text{None} \end{aligned}$$

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

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

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

3.3.3 Compositionality of the Semantics

Definition of the conjoin operator

constdefs

$$\begin{aligned} \text{same-length} &:: 'a \text{ par-confs} \Rightarrow ('a \text{ confs}) \text{ list} \Rightarrow \text{bool} \\ \text{same-length } c \ \text{clist} &\equiv (\forall i < \text{length } \text{clist}. \text{length}(\text{clist}!i) = \text{length } c) \end{aligned}$$

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

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

$$\begin{aligned} \text{compat-label} &:: 'a \text{ par-confs} \Rightarrow ('a \text{ confs}) \text{ list} \Rightarrow \text{bool} \\ \text{compat-label } c \ \text{clist} &\equiv (\forall j. \text{Suc } j < \text{length } c \longrightarrow \\ &\quad (c!j - \text{pc} \rightarrow c!\text{Suc } j \wedge (\exists i < \text{length } \text{clist}. (\text{clist}!i)!j - c \rightarrow (\text{clist}!i)! \text{Suc } j \wedge \end{aligned}$$

$$(\forall l < \text{length } \text{clist}. l \neq i \longrightarrow (\text{clist}!l)!j -e\rightarrow (\text{clist}!l)! \text{Suc } j))) \vee \\ (c!j -pe\rightarrow c!\text{Suc } j \wedge (\forall i < \text{length } \text{clist}. (\text{clist}!i)!j -e\rightarrow (\text{clist}!i)! \text{Suc } j)))$$

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

Some previous lemmas

lemma *list-eq-if* [rule-format]:

$\forall \text{ys}. \text{xs} = \text{ys} \longrightarrow (\text{length } \text{xs} = \text{length } \text{ys}) \longrightarrow (\forall i < \text{length } \text{xs}. \text{xs}!i = \text{ys}!i)$
 $\langle \text{proof} \rangle$

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

lemma *nth-tl*: $\llbracket \text{ys}!0 = a; \text{ys} \neq [] \rrbracket \Longrightarrow \text{ys} = (a \# (\text{tl } \text{ys}))$
 $\langle \text{proof} \rangle$

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

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

lemma *seq-not-eq1*: $\text{Seq } c1 \text{ } c2 \neq c1$
 $\langle \text{proof} \rangle$

lemma *seq-not-eq2*: $\text{Seq } c1 \text{ } c2 \neq c2$
 $\langle \text{proof} \rangle$

lemma *if-not-eq1*: $\text{Cond } b \text{ } c1 \text{ } c2 \neq c1$
 $\langle \text{proof} \rangle$

lemma *if-not-eq2*: $\text{Cond } b \text{ } c1 \text{ } c2 \neq c2$
 $\langle \text{proof} \rangle$

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

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

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

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

$\langle \text{proof} \rangle$

lemma *prog-not-eq-in-par-ctran* [simp]: $\neg (P, s) -pc \rightarrow (P, t)$
 $\langle \text{proof} \rangle$

lemma *tl-in-cptn*: $\llbracket a \# xs \in \text{cptn}; xs \neq [] \rrbracket \implies xs \in \text{cptn}$
 $\langle \text{proof} \rangle$

lemma *tl-zero*[rule-format]:
 $P (ys! \text{Suc } j) \longrightarrow \text{Suc } j < \text{length } ys \longrightarrow ys \neq [] \longrightarrow P (tl(ys)!j)$
 $\langle \text{proof} \rangle$

3.3.4 The Semantics is Compositional

lemma *aux-if* [rule-format]:
 $\forall xs \ s \ \text{clist}. (\text{length } \text{clist} = \text{length } xs \wedge (\forall i < \text{length } xs. (xs!i, s) \# \text{clist}!i \in \text{cptn})$
 $\wedge ((xs, s) \# ys \propto \text{map } (\lambda i. (\text{fst } i, s) \# \text{snd } i)) (\text{zip } xs \ \text{clist}))$
 $\longrightarrow (xs, s) \# ys \in \text{par-cptn})$
 $\langle \text{proof} \rangle$

lemma *less-Suc-0* [iff]: $(n < \text{Suc } 0) = (n = 0)$
 $\langle \text{proof} \rangle$

lemma *aux-onlyif* [rule-format]: $\forall xs \ s. (xs, s) \# ys \in \text{par-cptn} \longrightarrow$
 $(\exists \text{clist}. (\text{length } \text{clist} = \text{length } xs) \wedge$
 $(xs, s) \# ys \propto \text{map } (\lambda i. (\text{fst } i, s) \# (\text{snd } i)) (\text{zip } xs \ \text{clist})) \wedge$
 $(\forall i < \text{length } xs. (xs!i, s) \# (\text{clist}!i) \in \text{cptn}))$
 $\langle \text{proof} \rangle$

lemma *one-iff-aux*: $xs \neq [] \implies (\forall ys. ((xs, s) \# ys \in \text{par-cptn}) =$
 $(\exists \text{clist}. \text{length } \text{clist} = \text{length } xs \wedge$
 $((xs, s) \# ys \propto \text{map } (\lambda i. (\text{fst } i, s) \# (\text{snd } i)) (\text{zip } xs \ \text{clist})) \wedge$
 $(\forall i < \text{length } xs. (xs!i, s) \# (\text{clist}!i) \in \text{cptn}))) =$
 $(\text{par-cp } (xs) \ s = \{c. \exists \text{clist}. (\text{length } \text{clist}) = (\text{length } xs) \wedge$
 $(\forall i < \text{length } \text{clist}. (\text{clist}!i) \in \text{cp}(xs!i) \ s) \wedge c \propto \text{clist}\})$
 $\langle \text{proof} \rangle$

theorem *one*: $xs \neq [] \implies$
 $\text{par-cp } xs \ s = \{c. \exists \text{clist}. (\text{length } \text{clist}) = (\text{length } xs) \wedge$
 $(\forall i < \text{length } \text{clist}. (\text{clist}!i) \in \text{cp}(xs!i) \ s) \wedge c \propto \text{clist}\}$
 $\langle \text{proof} \rangle$

end

3.4 The Proof System

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

3.4.1 Proof System for Component Programs

declare *Un-subset-iff* [*iff del*]
declare *Cons-eq-map-conv*[*iff*]

constdefs

stable :: 'a set \Rightarrow ('a \times 'a) set \Rightarrow bool
stable $\equiv \lambda f g. (\forall x y. x \in f \longrightarrow (x, y) \in g \longrightarrow y \in f)$

inductive

rghoare :: ['a com, 'a set, ('a \times 'a) set, ('a \times 'a) set, 'a set] \Rightarrow bool
 $(\vdash - \text{sat } [-, -, -, -] [60, 0, 0, 0, 0] \ 45)$

where

Basic: $\llbracket pre \subseteq \{s. f \ s \in post\}; \{(s, t). s \in pre \wedge (t = f \ s \vee t = s)\} \subseteq guar; \text{stable } pre \text{ rely}; \text{stable } post \text{ rely} \rrbracket$
 $\Longrightarrow \vdash \text{Basic } f \text{ sat } [pre, \text{rely}, guar, post]$

| *Seq*: $\llbracket \vdash P \text{ sat } [pre, \text{rely}, guar, mid]; \vdash Q \text{ sat } [mid, \text{rely}, guar, post] \rrbracket$
 $\Longrightarrow \vdash \text{Seq } P \ Q \text{ sat } [pre, \text{rely}, guar, post]$

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

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

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

| *Conseq*: $\llbracket pre \subseteq pre'; \text{rely} \subseteq \text{rely}'; guar' \subseteq guar; post' \subseteq post; \vdash P \text{ sat } [pre', \text{rely}', guar', post'] \rrbracket$
 $\Longrightarrow \vdash P \text{ sat } [pre, \text{rely}, guar, post]$

constdefs

Pre :: 'a rformula \Rightarrow 'a set
Pre $x \equiv \text{fst}(\text{snd } x)$
Post :: 'a rformula \Rightarrow 'a set
Post $x \equiv \text{snd}(\text{snd}(\text{snd}(\text{snd } x)))$
Rely :: 'a rformula \Rightarrow ('a \times 'a) set
Rely $x \equiv \text{fst}(\text{snd}(\text{snd } x))$
Guar :: 'a rformula \Rightarrow ('a \times 'a) set
Guar $x \equiv \text{fst}(\text{snd}(\text{snd}(\text{snd } x)))$
Com :: 'a rformula \Rightarrow 'a com
Com $x \equiv \text{fst } x$

3.4.2 Proof System for Parallel Programs

types $'a \text{ par-rgformula} = ('a \text{ rgformula}) \text{ list} \times 'a \text{ set} \times ('a \times 'a) \text{ set} \times ('a \times 'a) \text{ set} \times 'a \text{ set}$

inductive

$\text{par-rghoare} :: ('a \text{ rgformula}) \text{ list} \Rightarrow 'a \text{ set} \Rightarrow ('a \times 'a) \text{ set} \Rightarrow ('a \times 'a) \text{ set} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$

$(\vdash - \text{SAT } [-, -, -, -] [60, 0, 0, 0, 0] 45)$

where

Parallel:

$\llbracket \forall i < \text{length } xs. \text{rely} \cup (\bigcup j \in \{j. j < \text{length } xs \wedge j \neq i\}. \text{Guar}(xs!j)) \subseteq \text{Rely}(xs!i);$
 $(\bigcup j \in \{j. j < \text{length } xs\}. \text{Guar}(xs!j)) \subseteq \text{guar};$
 $\text{pre} \subseteq (\bigcap i \in \{i. i < \text{length } xs\}. \text{Pre}(xs!i));$
 $(\bigcap i \in \{i. i < \text{length } xs\}. \text{Post}(xs!i)) \subseteq \text{post};$
 $\forall i < \text{length } xs. \vdash \text{Com}(xs!i) \text{ sat } [\text{Pre}(xs!i), \text{Rely}(xs!i), \text{Guar}(xs!i), \text{Post}(xs!i)] \rrbracket$
 $\implies \vdash xs \text{ SAT } [\text{pre}, \text{rely}, \text{guar}, \text{post}]$

3.5 Soundness

Some previous lemmas

lemma *tl-of-assum-in-assum*:

$(P, s) \# (P, t) \# xs \in \text{assum}(\text{pre}, \text{rely}) \implies \text{stable pre rely}$
 $\implies (P, t) \# xs \in \text{assum}(\text{pre}, \text{rely})$
 $\langle \text{proof} \rangle$

lemma *etran-in-comm*:

$(P, t) \# xs \in \text{comm}(\text{guar}, \text{post}) \implies (P, s) \# (P, t) \# xs \in \text{comm}(\text{guar}, \text{post})$
 $\langle \text{proof} \rangle$

lemma *ctran-in-comm*:

$\llbracket (s, s) \in \text{guar}; (Q, s) \# xs \in \text{comm}(\text{guar}, \text{post}) \rrbracket$
 $\implies (P, s) \# (Q, s) \# xs \in \text{comm}(\text{guar}, \text{post})$
 $\langle \text{proof} \rangle$

lemma *takecptn-is-cptn* [rule-format, elim!]:

$\forall j. c \in \text{cptn} \longrightarrow \text{take } (\text{Suc } j) \ c \in \text{cptn}$
 $\langle \text{proof} \rangle$

lemma *dropcptn-is-cptn* [rule-format, elim!]:

$\forall j < \text{length } c. c \in \text{cptn} \longrightarrow \text{drop } j \ c \in \text{cptn}$
 $\langle \text{proof} \rangle$

lemma *takepar-cptn-is-par-cptn* [rule-format, elim]:

$\forall j. c \in \text{par-cptn} \longrightarrow \text{take } (\text{Suc } j) \ c \in \text{par-cptn}$
 $\langle \text{proof} \rangle$

lemma *droppar-cptn-is-par-cptn* [rule-format]:

$\forall j < \text{length } c. c \in \text{par-cptn} \longrightarrow \text{drop } j \ c \in \text{par-cptn}$

$\langle \text{proof} \rangle$

lemma *tl-of-cptn-is-cptn*: $\llbracket x \# xs \in \text{cptn}; xs \neq [] \rrbracket \implies xs \in \text{cptn}$
 $\langle \text{proof} \rangle$

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

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

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

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

lemma *etran-or-ctran2-disjI1*:
 $\llbracket x \in \text{cptn}; \text{Suc } i < \text{length } x; x!i -c \longrightarrow x!\text{Suc } i \rrbracket \implies \neg x!i -e \longrightarrow x!\text{Suc } i$
 $\langle \text{proof} \rangle$

lemma *etran-or-ctran2-disjI2*:
 $\llbracket x \in \text{cptn}; \text{Suc } i < \text{length } x; x!i -e \longrightarrow x!\text{Suc } i \rrbracket \implies \neg x!i -c \longrightarrow x!\text{Suc } i$
 $\langle \text{proof} \rangle$

lemma *not-ctran-None2* [rule-format]:
 $\llbracket (\text{None}, s) \# xs \in \text{cptn}; i < \text{length } xs \rrbracket \implies \neg ((\text{None}, s) \# xs)!i -c \longrightarrow xs!i$
 $\langle \text{proof} \rangle$

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

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

3.5.1 Soundness of the System for Component Programs

Soundness of the Basic rule

lemma *unique-ctran-Basic* [rule-format]:

$$\begin{aligned} & \forall s \ i. \ x \in \text{cptn} \longrightarrow x ! 0 = (\text{Some } (\text{Basic } f), s) \longrightarrow \\ & \quad \text{Suc } i < \text{length } x \longrightarrow x!i \text{ --c--> } x!\text{Suc } i \longrightarrow \\ & \quad (\forall j. \ \text{Suc } j < \text{length } x \longrightarrow i \neq j \longrightarrow x!j \text{ --e--> } x!\text{Suc } j) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *exists-ctran-Basic-None* [rule-format]:

$$\begin{aligned} & \forall s \ i. \ x \in \text{cptn} \longrightarrow x ! 0 = (\text{Some } (\text{Basic } f), s) \\ & \longrightarrow i < \text{length } x \longrightarrow \text{fst}(x!i) = \text{None} \longrightarrow (\exists j < i. \ x!j \text{ --c--> } x!\text{Suc } j) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *Basic-sound*:

$$\begin{aligned} & \llbracket \text{pre} \subseteq \{s. f \ s \in \text{post}\}; \{(s, t). s \in \text{pre} \wedge t = f \ s\} \subseteq \text{guar}; \\ & \quad \text{stable pre rely}; \text{stable post rely} \rrbracket \\ & \implies \models \text{Basic } f \text{ sat } [\text{pre}, \text{rely}, \text{guar}, \text{post}] \\ & \langle \text{proof} \rangle \end{aligned}$$

Soundness of the Await rule

lemma *unique-ctran-Await* [rule-format]:

$$\begin{aligned} & \forall s \ i. \ x \in \text{cptn} \longrightarrow x ! 0 = (\text{Some } (\text{Await } b \ c), s) \longrightarrow \\ & \quad \text{Suc } i < \text{length } x \longrightarrow x!i \text{ --c--> } x!\text{Suc } i \longrightarrow \\ & \quad (\forall j. \ \text{Suc } j < \text{length } x \longrightarrow i \neq j \longrightarrow x!j \text{ --e--> } x!\text{Suc } j) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *exists-ctran-Await-None* [rule-format]:

$$\begin{aligned} & \forall s \ i. \ x \in \text{cptn} \longrightarrow x ! 0 = (\text{Some } (\text{Await } b \ c), s) \\ & \longrightarrow i < \text{length } x \longrightarrow \text{fst}(x!i) = \text{None} \longrightarrow (\exists j < i. \ x!j \text{ --c--> } x!\text{Suc } j) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *Star-imp-cptn*:

$$\begin{aligned} & (P, s) \text{ --c*--> } (R, t) \implies \exists l \in \text{cp } P \ s. \ (\text{last } l) = (R, t) \\ & \wedge (\forall i. \ \text{Suc } i < \text{length } l \longrightarrow l!i \text{ --c--> } l!\text{Suc } i) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *Await-sound*:

$$\begin{aligned} & \llbracket \text{stable pre rely}; \text{stable post rely}; \\ & \quad \forall V. \vdash P \text{ sat } [\text{pre} \cap b \cap \{s. s = V\}, \{(s, t). s = t\}, \\ & \quad \quad \text{UNIV}, \{s. (V, s) \in \text{guar}\} \cap \text{post}] \wedge \\ & \quad \models P \text{ sat } [\text{pre} \cap b \cap \{s. s = V\}, \{(s, t). s = t\}, \\ & \quad \quad \text{UNIV}, \{s. (V, s) \in \text{guar}\} \cap \text{post}] \rrbracket \\ & \implies \models \text{Await } b \ P \text{ sat } [\text{pre}, \text{rely}, \text{guar}, \text{post}] \\ & \langle \text{proof} \rangle \end{aligned}$$

Soundness of the Conditional rule

lemma *Cond-sound*:

$\llbracket \text{stable } pre \text{ rely}; \models P1 \text{ sat } [pre \cap b, \text{rely}, \text{guar}, \text{post}];$
 $\models P2 \text{ sat } [pre \cap -b, \text{rely}, \text{guar}, \text{post}]; \forall s. (s, s) \in \text{guar} \rrbracket$
 $\implies \models (\text{Cond } b \ P1 \ P2) \text{ sat } [pre, \text{rely}, \text{guar}, \text{post}]$
 $\langle \text{proof} \rangle$

Soundness of the Sequential rule

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

lemma *last-lift-not-None*: $\text{fst } ((\text{lift } Q) ((x \# xs)!(\text{length } xs))) \neq \text{None}$
 $\langle \text{proof} \rangle$

declare *map-eq-Cons-conv* [*simp del*] *Cons-eq-map-conv* [*simp del*]

lemma *Seq-sound1* [*rule-format*]:

$x \in \text{cptn-mod} \implies \forall s \ P. x ! 0 = (\text{Some } (\text{Seq } P \ Q), s) \longrightarrow$
 $(\forall i < \text{length } x. \text{fst}(x!i) \neq \text{Some } Q) \longrightarrow$
 $(\exists xs \in \text{cp } (\text{Some } P) \ s. x = \text{map } (\text{lift } Q) \ xs)$
 $\langle \text{proof} \rangle$

declare *map-eq-Cons-conv* [*simp del*] *Cons-eq-map-conv* [*simp del*]

lemma *Seq-sound2* [*rule-format*]:

$x \in \text{cptn} \implies \forall s \ P \ i. x ! 0 = (\text{Some } (\text{Seq } P \ Q), s) \longrightarrow i < \text{length } x$
 $\longrightarrow \text{fst}(x!i) = \text{Some } Q \longrightarrow$
 $(\forall j < i. \text{fst}(x!j) \neq (\text{Some } Q)) \longrightarrow$
 $(\exists xs \ ys. xs \in \text{cp } (\text{Some } P) \ s \wedge \text{length } xs = \text{Suc } i$
 $\wedge ys \in \text{cp } (\text{Some } Q) \ (\text{snd}(xs ! i)) \wedge x = (\text{map } (\text{lift } Q) \ xs) @ \text{tl } ys)$
 $\langle \text{proof} \rangle$

lemma *last-lift-not-None2*: $\text{fst } ((\text{lift } Q) (\text{last } (x \# xs))) \neq \text{None}$
 $\langle \text{proof} \rangle$

lemma *Seq-sound*:

$\llbracket \models P \text{ sat } [pre, \text{rely}, \text{guar}, \text{mid}]; \models Q \text{ sat } [\text{mid}, \text{rely}, \text{guar}, \text{post}] \rrbracket$
 $\implies \models \text{Seq } P \ Q \text{ sat } [pre, \text{rely}, \text{guar}, \text{post}]$
 $\langle \text{proof} \rangle$

Soundness of the While rule

lemma *last-append* [*rule-format*]:

$\forall xs. ys \neq [] \longrightarrow ((xs @ ys)!(\text{length } (xs @ ys) - (\text{Suc } 0))) = (ys!(\text{length } ys - (\text{Suc } 0)))$
 $\langle \text{proof} \rangle$

lemma *assum-after-body*:

$\llbracket \models P \text{ sat } [pre \cap b, \text{rely}, \text{guar}, \text{pre}];$
 $(\text{Some } P, s) \# xs \in \text{cptn-mod}; \text{fst } (\text{last } ((\text{Some } P, s) \# xs)) = \text{None}; s \in b;$
 $(\text{Some } (\text{While } b \ P), s) \# (\text{Some } (\text{Seq } P \ (\text{While } b \ P)), s) \#$

$\text{map } (\text{lift } (\text{While } b \ P)) \ xs \ @ \ ys \in \text{assum } (pre, \text{rely})$
 $\implies (\text{Some } (\text{While } b \ P), \text{snd } (\text{last } ((\text{Some } P, s) \# xs))) \# ys \in \text{assum } (pre, \text{rely})$
 $\langle \text{proof} \rangle$

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

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

Soundness of the Rule of Consequence

lemma *Conseq-sound*:
 $\llbracket pre \subseteq pre'; \text{rely} \subseteq \text{rely}'; guar' \subseteq guar; post' \subseteq post;$
 $\models P \text{ sat } [pre', \text{rely}', guar', post'] \rrbracket$
 $\implies \models P \text{ sat } [pre, \text{rely}, guar, post]$
 $\langle \text{proof} \rangle$

Soundness of the system for sequential component programs

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

3.5.2 Soundness of the System for Parallel Programs

constdefs
 $\text{ParallelCom} :: ('a \ \text{rgformula}) \ \text{list} \Rightarrow 'a \ \text{par-com}$
 $\text{ParallelCom } Ps \equiv \text{map } (\text{Some} \circ \text{fst}) \ Ps$

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

lemma three [rule-format]:

$$\begin{aligned}
& \llbracket xs \neq []; \forall i < \text{length } xs. \text{rely} \cup (\bigcup j \in \{j. j < \text{length } xs \wedge j \neq i\}. \text{Guar } (xs ! j)) \\
& \subseteq \text{Rely } (xs ! i); \\
& \text{pre} \subseteq (\bigcap i \in \{i. i < \text{length } xs\}. \text{Pre } (xs ! i)); \\
& \forall i < \text{length } xs. \\
& \quad \models \text{Com } (xs ! i) \text{ sat } [\text{Pre } (xs ! i), \text{Rely } (xs ! i), \text{Guar } (xs ! i), \text{Post } (xs ! i)]; \\
& \quad \text{length } xs = \text{length } \text{clist}; x \in \text{par-cp } (\text{ParallelCom } xs) \ s; x \in \text{par-assum}(\text{pre}, \text{rely}); \\
& \quad \forall i < \text{length } \text{clist}. \text{clist}!i \in \text{cp } (\text{Some}(\text{Com}(xs!i))) \ s; x \propto \text{clist} \rrbracket \\
& \implies \forall j \ i. i < \text{length } \text{clist} \wedge \text{Suc } j < \text{length } x \longrightarrow (\text{clist}!i!j) -e\rightarrow (\text{clist}!i!\text{Suc } j) \\
& \longrightarrow (\text{snd}(\text{clist}!i!j), \text{snd}(\text{clist}!i!\text{Suc } j)) \in \text{rely} \cup (\bigcup j \in \{j. j < \text{length } xs \wedge j \neq i\}. \\
& \text{Guar } (xs ! j)) \\
& \langle \text{proof} \rangle
\end{aligned}$$

lemma four:

$$\begin{aligned}
& \llbracket xs \neq []; \forall i < \text{length } xs. \text{rely} \cup (\bigcup j \in \{j. j < \text{length } xs \wedge j \neq i\}. \text{Guar } (xs ! j)) \\
& \subseteq \text{Rely } (xs ! i); \\
& (\bigcup j \in \{j. j < \text{length } xs\}. \text{Guar } (xs ! j)) \subseteq \text{guar}; \\
& \text{pre} \subseteq (\bigcap i \in \{i. i < \text{length } xs\}. \text{Pre } (xs ! i)); \\
& \forall i < \text{length } xs. \\
& \quad \models \text{Com } (xs ! i) \text{ sat } [\text{Pre } (xs ! i), \text{Rely } (xs ! i), \text{Guar } (xs ! i), \text{Post } (xs ! i)]; \\
& \quad x \in \text{par-cp } (\text{ParallelCom } xs) \ s; x \in \text{par-assum } (\text{pre}, \text{rely}); \text{Suc } i < \text{length } x; \\
& \quad x ! i -pc\rightarrow x ! \text{Suc } i \rrbracket \\
& \implies (\text{snd } (x ! i), \text{snd } (x ! \text{Suc } i)) \in \text{guar} \\
& \langle \text{proof} \rangle
\end{aligned}$$

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

$\langle \text{proof} \rangle$

lemma five:

$$\begin{aligned}
& \llbracket xs \neq []; \forall i < \text{length } xs. \text{rely} \cup (\bigcup j \in \{j. j < \text{length } xs \wedge j \neq i\}. \text{Guar } (xs ! j)) \\
& \subseteq \text{Rely } (xs ! i); \\
& \text{pre} \subseteq (\bigcap i \in \{i. i < \text{length } xs\}. \text{Pre } (xs ! i)); \\
& (\bigcap i \in \{i. i < \text{length } xs\}. \text{Post } (xs ! i)) \subseteq \text{post}; \\
& \forall i < \text{length } xs. \\
& \quad \models \text{Com } (xs ! i) \text{ sat } [\text{Pre } (xs ! i), \text{Rely } (xs ! i), \text{Guar } (xs ! i), \text{Post } (xs ! i)]; \\
& \quad x \in \text{par-cp } (\text{ParallelCom } xs) \ s; x \in \text{par-assum } (\text{pre}, \text{rely}); \\
& \quad \text{All-None } (\text{fst } (\text{last } x)) \rrbracket \implies \text{snd } (\text{last } x) \in \text{post} \\
& \langle \text{proof} \rangle
\end{aligned}$$

lemma ParallelEmpty [rule-format]:

$$\begin{aligned}
& \forall i \ s. x \in \text{par-cp } (\text{ParallelCom } []) \ s \longrightarrow \\
& \quad \text{Suc } i < \text{length } x \longrightarrow (x ! i, x ! \text{Suc } i) \notin \text{par-ctran} \\
& \langle \text{proof} \rangle
\end{aligned}$$

theorem par-rgsound:

$$\begin{aligned}
& \vdash c \text{ SAT } [\text{pre}, \text{rely}, \text{guar}, \text{post}] \implies \\
& \quad \models (\text{ParallelCom } c) \text{ SAT } [\text{pre}, \text{rely}, \text{guar}, \text{post}] \\
& \langle \text{proof} \rangle
\end{aligned}$$

end

3.6 Concrete Syntax

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

syntax

-Assign :: $idt \Rightarrow 'b \Rightarrow 'a\ com$ (($'- := / -$) [70, 65] 61)
 -skip :: $'a\ com$ (SKIP)
 -Seq :: $'a\ com \Rightarrow 'a\ com \Rightarrow 'a\ com$ (($-;; / -$) [60, 61] 60)
 -Cond :: $'a\ bexp \Rightarrow 'a\ com \Rightarrow 'a\ com \Rightarrow 'a\ com$ (($0IF - / THEN - / ELSE - / FI$) [0, 0, 0] 61)
 -Cond2 :: $'a\ bexp \Rightarrow 'a\ com \Rightarrow 'a\ com$ (($0IF - THEN - FI$) [0, 0] 56)
 -While :: $'a\ bexp \Rightarrow 'a\ com \Rightarrow 'a\ com$ (($0WHILE - / DO - / OD$) [0, 0] 61)
 -Await :: $'a\ bexp \Rightarrow 'a\ com \Rightarrow 'a\ com$ (($0AWAIT - / THEN - / END$) [0, 0] 61)
 -Atom :: $'a\ com \Rightarrow 'a\ com$ ((($\langle - \rangle$) 61)
 -Wait :: $'a\ bexp \Rightarrow 'a\ com$ (($0WAIT - END$) 61)

translations

$x := a \rightarrow Basic \ll '(-update-name\ x\ (\lambda-. a)) \gg$
 $SKIP \rightleftharpoons Basic\ id$
 $c1;; c2 \rightleftharpoons Seq\ c1\ c2$
 $IF\ b\ THEN\ c1\ ELSE\ c2\ FI \rightarrow Cond\ .\{b\}.\ c1\ c2$
 $IF\ b\ THEN\ c\ FI \rightleftharpoons IF\ b\ THEN\ c\ ELSE\ SKIP\ FI$
 $WHILE\ b\ DO\ c\ OD \rightarrow While\ .\{b\}.\ c$
 $AWAIT\ b\ THEN\ c\ END \rightleftharpoons Await\ .\{b\}.\ c$
 $\langle c \rangle \rightleftharpoons AWAIT\ True\ THEN\ c\ END$
 $WAIT\ b\ END \rightleftharpoons AWAIT\ b\ THEN\ SKIP\ END$

nonterminals

prgs

syntax

-PAR :: $prgs \Rightarrow 'a$ (COBEGIN//-/COEND 60)
 -prg :: $'a \Rightarrow prgs$ (- 57)
 -prgs :: $['a, prgs] \Rightarrow prgs$ (-//||/- [60, 57] 57)

translations

-prg $a \rightarrow [a]$
 -prgs $a\ ps \rightarrow a \# ps$
 -PAR $ps \rightarrow ps$

syntax

-prg-scheme :: $['a, 'a, 'a, 'a] \Rightarrow prgs$ (SCHEME [- ≤ - < -] - [0, 0, 0, 60] 57)

translations

-prg-scheme $j \ i \ k \ c \rightleftharpoons (\text{map } (\lambda i. \ c) \ [j..<k])$

Translations for variables before and after a transition:

syntax

-before $:: id \Rightarrow 'a \ (^{\circ}-)$

-after $:: id \Rightarrow 'a \ (^{\text{a}}-)$

translations

$^{\circ}x \rightleftharpoons x \ 'fst$

$^{\text{a}}x \rightleftharpoons x \ 'snd$

$\langle ML \rangle$

end

3.7 Examples

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

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

3.7.1 Set Elements of an Array to Zero

lemma *le-less-trans2*: $\llbracket (j::\text{nat}) < k; i \leq j \rrbracket \Longrightarrow i < k$

$\langle \text{proof} \rangle$

lemma *add-le-less-mono*: $\llbracket (a::\text{nat}) < c; b \leq d \rrbracket \Longrightarrow a + b < c + d$

$\langle \text{proof} \rangle$

record *Example1* =

A :: *nat list*

lemma *Example1*:

$\vdash \text{COBEGIN}$

SCHEME $[0 \leq i < n]$

$('A := 'A \ [i := 0],$

$\llbracket n < \text{length } 'A \rrbracket,$

$\llbracket \text{length } ^{\circ}A = \text{length } ^{\text{a}}A \wedge ^{\circ}A ! i = ^{\text{a}}A ! i \rrbracket,$

$\llbracket \text{length } ^{\circ}A = \text{length } ^{\text{a}}A \wedge (\forall j < n. i \neq j \longrightarrow ^{\circ}A ! j = ^{\text{a}}A ! j) \rrbracket,$

$\llbracket 'A ! i = 0 \rrbracket)$

COEND

SAT $\llbracket \llbracket n < \text{length } 'A \rrbracket, \llbracket ^{\circ}A = ^{\text{a}}A \rrbracket, \llbracket \text{True} \rrbracket, \llbracket \forall i < n. 'A ! i = 0 \rrbracket \rrbracket$

$\langle \text{proof} \rangle$

lemma *Example1-parameterized*:

$k < t \Longrightarrow$

$\vdash \text{COBEGIN}$

$SCHEME \ [k*n \leq i < (Suc\ k)*n] \ (\ 'A := 'A[i := 0],$
 $\{ \{ t*n < length\ 'A \},$
 $\{ \{ t*n < length\ ^oA \wedge length\ ^oA = length\ ^aA \wedge ^oA!i = ^aA!i \},$
 $\{ \{ t*n < length\ ^oA \wedge length\ ^oA = length\ ^aA \wedge (\forall j < length\ ^oA . i \neq j \longrightarrow ^oA!j =$
 $^aA!j) \},$
 $\{ \ 'A!i = 0 \})$
 $COEND$
 $SAT \ [\{ \{ t*n < length\ 'A \},$
 $\{ \{ t*n < length\ ^oA \wedge length\ ^oA = length\ ^aA \wedge (\forall i < n . ^oA!(k*n+i) = ^aA!(k*n+i)) \},$
 $\{ \{ t*n < length\ ^oA \wedge length\ ^oA = length\ ^aA \wedge$
 $(\forall i < length\ ^oA . (i < k*n \longrightarrow ^oA!i = ^aA!i) \wedge ((Suc\ k)*n \leq i \longrightarrow ^oA!i =$
 $^aA!i)) \},$
 $\{ \{ \forall i < n . 'A!(k*n+i) = 0 \} \}]$
 $\langle proof \rangle$

3.7.2 Increment a Variable in Parallel

Two components

record *Example2* =

$x :: nat$
 $c-0 :: nat$
 $c-1 :: nat$

lemma *Example2*:

$\vdash \ COBEGIN$
 $(\langle \ 'x := 'x+1;; \ 'c-0 := 'c-0 + 1 \rangle,$
 $\{ \{ 'x = 'c-0 + 'c-1 \wedge 'c-0 = 0 \},$
 $\{ \{ ^o c-0 = ^a c-0 \wedge$
 $(^o x = ^o c-0 + ^o c-1$
 $\longrightarrow ^a x = ^a c-0 + ^a c-1) \},$
 $\{ \{ ^o c-1 = ^a c-1 \wedge$
 $(^o x = ^o c-0 + ^o c-1$
 $\longrightarrow ^a x = ^a c-0 + ^a c-1) \},$
 $\{ \{ 'x = 'c-0 + 'c-1 \wedge 'c-0 = 1 \} \})$
 \parallel
 $(\langle \ 'x := 'x+1;; \ 'c-1 := 'c-1+1 \rangle,$
 $\{ \{ 'x = 'c-0 + 'c-1 \wedge 'c-1 = 0 \},$
 $\{ \{ ^o c-1 = ^a c-1 \wedge$
 $(^o x = ^o c-0 + ^o c-1$
 $\longrightarrow ^a x = ^a c-0 + ^a c-1) \},$
 $\{ \{ ^o c-0 = ^a c-0 \wedge$
 $(^o x = ^o c-0 + ^o c-1$
 $\longrightarrow ^a x = ^a c-0 + ^a c-1) \},$
 $\{ \{ 'x = 'c-0 + 'c-1 \wedge 'c-1 = 1 \} \})$
 $COEND$
 $SAT \ [\{ \{ 'x = 0 \wedge 'c-0 = 0 \wedge 'c-1 = 0 \},$
 $\{ \{ ^o x = ^a x \wedge ^o c-0 = ^a c-0 \wedge ^o c-1 = ^a c-1 \},$
 $\{ \{ True \},$

$\{\text{' }x=2\}$
 $\langle \text{proof} \rangle$

Parameterized

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

lemma *Example2-lemma2-aux2*:
 $j \leq s \implies (\sum_{i::\text{nat}=0..<j.} (b \ (s:=t)) \ i) = (\sum_{i=0..<j.} b \ i)$
 $\langle \text{proof} \rangle$

lemma *Example2-lemma2*:
 $\llbracket j < n; b \ j = 0 \rrbracket \implies \text{Suc } (\sum_{i::\text{nat}=0..<n.} b \ i) = (\sum_{i=0..<n.} (b \ (j := \text{Suc } 0)) \ i)$
 $\langle \text{proof} \rangle$

lemma *Example2-lemma2-Suc0*: $\llbracket j < n; b \ j = 0 \rrbracket \implies$
 $\text{Suc } (\sum_{i::\text{nat}=0..<n.} b \ i) = (\sum_{i=0..<n.} (b \ (j := \text{Suc } 0)) \ i)$
 $\langle \text{proof} \rangle$

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

lemma *Example2-parameterized*: $0 < n \implies$
 $\vdash \text{COBEGIN SCHEME } [0 \leq i < n]$
 $(\langle \text{' }y := \text{' }y + 1;; \text{' }C := \text{' }C \ (i := 1) \rangle,$
 $\{\text{' }y = (\sum_{i=0..<n.} \text{' }C \ i) \wedge \text{' }C \ i = 0\},$
 $\{\text{' }C \ i = {}^a C \ i \wedge$
 $({}^o y = (\sum_{i=0..<n.} {}^o C \ i) \longrightarrow {}^a y = (\sum_{i=0..<n.} {}^a C \ i))\},$
 $\{(\forall j < n. i \neq j \longrightarrow {}^o C \ j = {}^a C \ j) \wedge$
 $({}^o y = (\sum_{i=0..<n.} {}^o C \ i) \longrightarrow {}^a y = (\sum_{i=0..<n.} {}^a C \ i))\},$
 $\{\text{' }y = (\sum_{i=0..<n.} \text{' }C \ i) \wedge \text{' }C \ i = 1\})$
 COEND
 $\text{SAT } [\{\text{' }y = 0 \wedge (\sum_{i=0..<n.} \text{' }C \ i) = 0\}, \{\text{' }C = {}^a C \wedge {}^o y = {}^a y\}, \{\text{True}\}, \{\text{' }y = n\}]$
 $\langle \text{proof} \rangle$

3.7.3 Find Least Element

A previous lemma:

lemma *mod-aux*: $\llbracket i < (n :: \text{nat}); a \bmod n = i; j < a + n; j \bmod n = i; a < j \rrbracket$
 $\implies \text{False}$
 $\langle \text{proof} \rangle$

record *Example3* =
 $X :: \text{nat} \Rightarrow \text{nat}$
 $Y :: \text{nat} \Rightarrow \text{nat}$

lemma *Example3*: $m \bmod n = 0 \implies$
 \vdash COBEGIN
SCHEME $[0 \leq i < n]$
(WHILE $(\forall j < n. 'X\ i < 'Y\ j)$ DO
IF $P(B!('X\ i))$ THEN $'Y := 'Y\ (i := 'X\ i)$
ELSE $'X := 'X\ (i := ('X\ i) + n)$ FI
OD,
 $\{\{('X\ i) \bmod n = i \wedge (\forall j < 'X\ i. j \bmod n = i \longrightarrow \neg P(B!j)) \wedge ('Y\ i < m \longrightarrow P(B!('Y\ i)) \wedge 'Y\ i \leq m+i)\}\},$
 $\{\{(\forall j < n. i \neq j \longrightarrow {}^aY\ j \leq {}^oY\ j) \wedge {}^oX\ i = {}^aX\ i \wedge$
 ${}^oY\ i = {}^aY\ i\}\},$
 $\{\{(\forall j < n. i \neq j \longrightarrow {}^oX\ j = {}^aX\ j \wedge {}^oY\ j = {}^aY\ j) \wedge$
 ${}^aY\ i \leq {}^oY\ i\}\},$
 $\{\{('X\ i) \bmod n = i \wedge (\forall j < 'X\ i. j \bmod n = i \longrightarrow \neg P(B!j)) \wedge ('Y\ i < m \longrightarrow P(B!('Y\ i)) \wedge 'Y\ i \leq m+i) \wedge (\exists j < n. 'Y\ j \leq 'X\ i)\}\}$
COEND
SAT $[\{\{ \forall i < n. 'X\ i = i \wedge 'Y\ i = m+i \}, \{\{ {}^oX = {}^aX \wedge {}^oY = {}^aY \}\}, \{\{ True \}\},$
 $\{\{ \forall i < n. ('X\ i) \bmod n = i \wedge (\forall j < 'X\ i. j \bmod n = i \longrightarrow \neg P(B!j)) \wedge$
 $('Y\ i < m \longrightarrow P(B!('Y\ i)) \wedge 'Y\ i \leq m+i) \wedge (\exists j < n. 'Y\ j \leq 'X\ i)\}\}]$
 $\langle proof \rangle$

Same but with a list as auxiliary variable:

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

lemma *Example3-list*: $m \bmod n = 0 \implies \vdash$ (COBEGIN SCHEME $[0 \leq i < n]$
(WHILE $(\forall j < n. 'X!i < 'Y!j)$ DO
IF $P(B!('X!i))$ THEN $'Y := 'Y[i := 'X!i]$ ELSE $'X := 'X[i := ('X!i) + n]$ FI
OD,
 $\{\{n < length\ 'X \wedge n < length\ 'Y \wedge ('X!i) \bmod n = i \wedge (\forall j < 'X!i. j \bmod n = i \longrightarrow$
 $\neg P(B!j)) \wedge ('Y!i < m \longrightarrow P(B!('Y!i)) \wedge 'Y!i \leq m+i)\}\},$
 $\{\{(\forall j < n. i \neq j \longrightarrow {}^aY!j \leq {}^oY!j) \wedge {}^oX!i = {}^aX!i \wedge$
 ${}^oY!i = {}^aY!i \wedge length\ {}^oX = length\ {}^aX \wedge length\ {}^oY = length\ {}^aY\}\},$
 $\{\{(\forall j < n. i \neq j \longrightarrow {}^oX!j = {}^aX!j \wedge {}^oY!j = {}^aY!j) \wedge$
 ${}^aY!i \leq {}^oY!i \wedge length\ {}^oX = length\ {}^aX \wedge length\ {}^oY = length\ {}^aY\}\},$
 $\{\{('X!i) \bmod n = i \wedge (\forall j < 'X!i. j \bmod n = i \longrightarrow \neg P(B!j)) \wedge ('Y!i < m \longrightarrow P(B!('Y!i))$
 $\wedge 'Y!i \leq m+i) \wedge (\exists j < n. 'Y!j \leq 'X!i)\}\}$ COEND)
SAT $[\{\{n < length\ 'X \wedge n < length\ 'Y \wedge (\forall i < n. 'X!i = i \wedge 'Y!i = m+i)\}\},$
 $\{\{ {}^oX = {}^aX \wedge {}^oY = {}^aY \}\},$
 $\{\{ True \}\},$
 $\{\{ \forall i < n. ('X!i) \bmod n = i \wedge (\forall j < 'X!i. j \bmod n = i \longrightarrow \neg P(B!j)) \wedge$
 $('Y!i < m \longrightarrow P(B!('Y!i)) \wedge 'Y!i \leq m+i) \wedge (\exists j < n. 'Y!j \leq 'X!i)\}\}]$
 $\langle proof \rangle$

end

Bibliography

- [1] Leonor Prensa Nieto. *Verification of Parallel Programs with the Owicki-Gries and Rely-Guarantee Methods in Isabelle/HOL*. PhD thesis, Technische Universität München, 2002.
- [2] Tobias Nipkow and Leonor Prensa Nieto. Owicki/Gries in Isabelle/HOL. In J.-P. Finance, editor, *Fundamental Approaches to Software Engineering (FASE'99)*, volume 1577 of *LNCS*, pages 188–203. Springer, 1999.
- [3] Leonor Prensa Nieto. The Rely-Guarantee method in Isabelle/HOL. In P. Degano, editor, *European Symposium on Programming (ESOP'03)*, volume 2618, pages 348–362, 2003.
- [4] Leonor Prensa Nieto and Javier Esparza. Verifying single and multi-mutator garbage collectors with Owicki/Gries in Isabelle/HOL. In M. Nielsen and B. Rovan, editors, *Mathematical Foundations of Computer Science (MFCS 2000)*, volume 1893 of *LNCS*, pages 619–628. Springer-Verlag, 2000.