# Hoare Logic for Parallel Programs

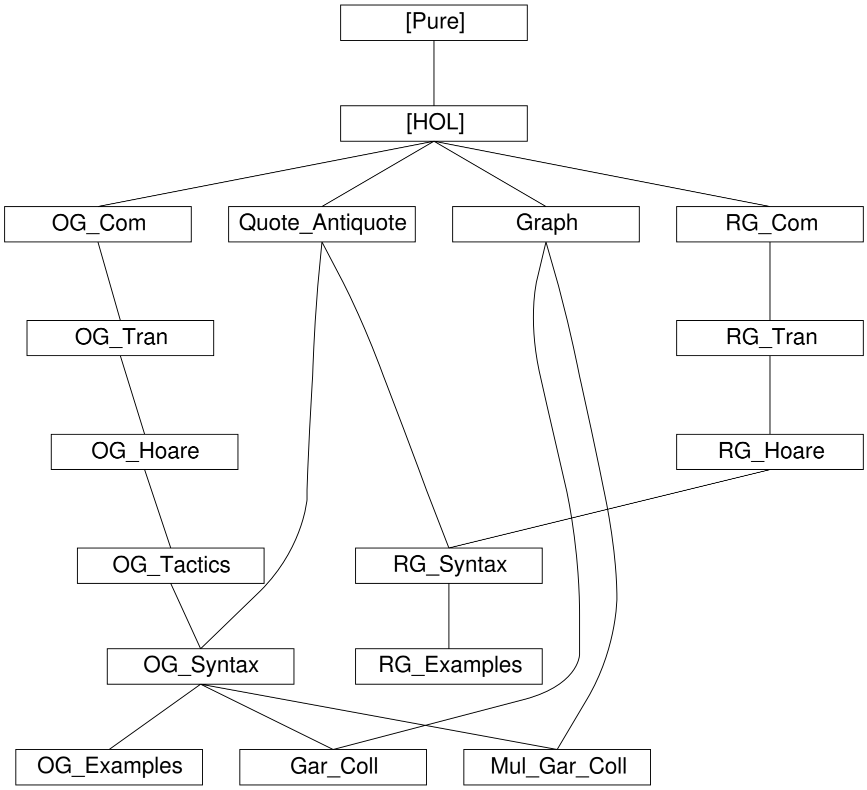
Leonor Prensa Nieto

1st October 2005

**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 [?].) 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.

# Contents


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

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

**end**

## 1.2 Operational Semantics

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

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

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

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

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

### 1.2.1 The Transition Relation

**consts**
$ann\text{-}transition :: (('a\ ann\text{-}com\text{-}op \times 'a) \times ('a\ ann\text{-}com\text{-}op \times 'a))\ set$
$transition :: (('a\ com \times 'a) \times ('a\ com \times 'a))\ set$

**syntax**
$\text{-}ann\text{-}transition :: ('a\ ann\text{-}com\text{-}op \times 'a) \Rightarrow ('a\ ann\text{-}com\text{-}op \times 'a) \Rightarrow bool$
$(\text{-}\ {-1\rightarrow}\ \text{-}[81,81]\ 100)$
$\text{-}ann\text{-}transition\text{-}n :: ('a\ ann\text{-}com\text{-}op \times 'a) \Rightarrow nat \Rightarrow ('a\ ann\text{-}com\text{-}op \times 'a)$
$\Rightarrow bool\ \ (\text{-}\ {-\text{-}\rightarrow}\ \text{-}[81,81]\ 100)$
$\text{-}ann\text{-}transition\text{-}* :: ('a\ ann\text{-}com\text{-}op \times 'a) \Rightarrow ('a\ ann\text{-}com\text{-}op \times 'a) \Rightarrow bool$
$(\text{-}\ {-*\rightarrow}\ \text{-}[81,81]\ 100)$

$\text{-}transition :: ('a\ com \times 'a) \Rightarrow ('a\ com \times 'a) \Rightarrow bool\ \ (\text{-}\ {-P1\rightarrow}\ \text{-}[81,81]\ 100)$
$\text{-}transition\text{-}n :: ('a\ com \times 'a) \Rightarrow nat \Rightarrow ('a\ com \times 'a) \Rightarrow bool$
$(\text{-}\ {-P\text{-}\rightarrow}\ \text{-}[81,81,81]\ 100)$
$\text{-}transition\text{-}* :: ('a\ com \times 'a) \Rightarrow ('a\ com \times 'a) \Rightarrow bool\ \ (\text{-}\ {-P*\rightarrow}\ \text{-}[81,81]\ 100)$

The corresponding syntax translations are:

**translations**
$con\text{-}0\ -1\rightarrow\ con\text{-}1 \rightleftharpoons (con\text{-}0,\ con\text{-}1) \in ann\text{-}transition$
$con\text{-}0\ -n\rightarrow\ con\text{-}1 \rightleftharpoons (con\text{-}0,\ con\text{-}1) \in ann\text{-}transition\hat{}n$
$con\text{-}0\ -*\rightarrow\ con\text{-}1 \rightleftharpoons (con\text{-}0,\ con\text{-}1) \in ann\text{-}transition^*$

$con\text{-}0\ -P1\rightarrow\ con\text{-}1 \rightleftharpoons (con\text{-}0,\ con\text{-}1) \in transition$
$con\text{-}0\ -Pn\rightarrow\ con\text{-}1 \rightleftharpoons (con\text{-}0,\ con\text{-}1) \in transition\hat{}n$
$con\text{-}0\ -P*\rightarrow\ con\text{-}1 \rightleftharpoons (con\text{-}0,\ con\text{-}1) \in transition^*$

**inductive** *ann-transition transition*
**intros**
$AnnBasic$: $(Some\ (AnnBasic\ r\ f),\ s)\ -1\rightarrow\ (None,\ f\ s)$

$AnnSeq1$: $(Some\ c0,\ s)\ -1\rightarrow\ (None,\ t) \Longrightarrow$
$(Some\ (AnnSeq\ c0\ c1),\ s)\ -1\rightarrow\ (Some\ c1,\ t)$
$AnnSeq2$: $(Some\ c0,\ s)\ -1\rightarrow\ (Some\ c2,\ t) \Longrightarrow$
$(Some\ (AnnSeq\ c0\ c1),\ s)\ -1\rightarrow\ (Some\ (AnnSeq\ c2\ c1),\ t)$

$AnnCond1T$: $s \in b \Longrightarrow (Some\ (AnnCond1\ r\ b\ c1\ c2),\ s)\ -1\rightarrow\ (Some\ c1,\ s)$
$AnnCond1F$: $s \notin b \Longrightarrow (Some\ (AnnCond1\ r\ b\ c1\ c2),\ s)\ -1\rightarrow\ (Some\ c2,\ s)$

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

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

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

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

$Basic$: $(Basic\ f,\ s)\ -P1\rightarrow\ (Parallel\ [],\ f\ s)$

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

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

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

**monos** *rtrancl-mono*

### 1.2.2 Definition of Semantics

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

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

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

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

**syntax** $\text{-}Omega :: {}'a\ com \quad (\Omega\ 63)$
**translations** $\Omega \rightleftharpoons While\ UNIV\ UNIV\ (Basic\ id)$

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

### Proofs

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

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

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

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

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

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

**lemma** *L3-5i*: $X \subseteq Y \Longrightarrow SEM\ c\ X \subseteq SEM\ c\ Y$

$\langle proof \rangle$

**lemma** *L3-5ii-lemma1*:
$[\![$ (*c1*, *s1*) $-P*\rightarrow$ (*Parallel Ts*, *s2*); *All-None Ts*;
(*c2*, *s2*) $-P*\rightarrow$ (*Parallel Ss*, *s3*); *All-None Ss* $]\!]$
$\Longrightarrow$ (*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*:
$[\![$(*Seq c1 c2*,*s*) $-P*\rightarrow$ (*Parallel Ts*,*t*); *All-None Ts*$]\!]$ $\Longrightarrow$
($\exists$ *y Rs*. (*c1*,*s*) $-P*\rightarrow$ (*Parallel Rs*,*y*) $\wedge$ *All-None Rs*
$\wedge$ (*c2*,*y*) $-P*\rightarrow$ (*Parallel Ts*,*t*))
$\langle proof \rangle$

**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*: $[\![$($\Omega$, *s*) $-P*\rightarrow$ (*Parallel Ts*, *t*); *All-None Ts* $]\!]$ $\Longrightarrow$ *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*)

⟨*proof*⟩

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

## 1.3 Validity of Correctness Formulas

**constdefs**
*com-validity* :: *'a assn* ⇒ *'a com* ⇒ *'a assn* ⇒ *bool* ((*3*∥= -// -//-) [*90*,*55*,*90*] *50*)
∥= *p c q* ≡ *SEM c p* ⊆ *q*

*ann-com-validity* :: *'a ann-com* ⇒ *'a assn* ⇒ *bool* (⊨ - - [*60*,*90*] *45*)
⊨ *c q* ≡ *ann-SEM c* (*pre c*) ⊆ *q*

**end**

## 1.4 The Proof System

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

**consts** *assertions* :: *'a ann-com* ⇒ (*'a assn*) *set*
**primrec**
*assertions* (*AnnBasic r f*) = {*r*}
*assertions* (*AnnSeq c1 c2*) = *assertions c1* ∪ *assertions c2*
*assertions* (*AnnCond1 r b c1 c2*) = {*r*} ∪ *assertions c1* ∪ *assertions c2*
*assertions* (*AnnCond2 r b c*) = {*r*} ∪ *assertions c*
*assertions* (*AnnWhile r b i c*) = {*r*, *i*} ∪ *assertions c*
*assertions* (*AnnAwait r b c*) = {*r*}

**consts** *atomics* :: *'a ann-com* ⇒ (*'a assn* × *'a com*) *set*
**primrec**
*atomics* (*AnnBasic r f*) = {(*r*, *Basic f*)}
*atomics* (*AnnSeq c1 c2*) = *atomics c1* ∪ *atomics c2*
*atomics* (*AnnCond1 r b c1 c2*) = *atomics c1* ∪ *atomics c2*
*atomics* (*AnnCond2 r b c*) = *atomics c*
*atomics* (*AnnWhile r b i c*) = *atomics c*
*atomics* (*AnnAwait r b c*) = {(*r* ∩ *b*, *c*)}

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

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

**constdefs** *interfree-aux* :: (*'a ann-com-op* × *'a assn* × *'a ann-com-op*) ⇒ *bool*
*interfree-aux* ≡ λ(*co*, *q*, *co'*). *co'*= *None* ∨
(∀ (*r*,*a*) ∈ *atomics* (*the co'*). ∥= (*q* ∩ *r*) *a q* ∧

$(co = None \lor (\forall p \in assertions\ (the\ co).\ \Vert= (p \cap r)\ a\ p)))$

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

**consts** *ann-hoare* :: $('a\ ann\text{-}com \times 'a\ assn)\ set$
**syntax** *-ann-hoare* :: $'a\ ann\text{-}com \Rightarrow 'a\ assn \Rightarrow bool\ \ ((2\vdash\ \text{-}//\ \text{-})\ [60,90]\ 45)$
**translations** $\vdash c\ q \rightleftharpoons (c,\ q) \in ann\text{-}hoare$

**consts** *oghoare* :: $('a\ assn \times 'a\ com \times 'a\ assn)\ set$
**syntax** *-oghoare* :: $'a\ assn \Rightarrow 'a\ com \Rightarrow 'a\ assn \Rightarrow bool\ \ ((3\Vert-\ \text{-}//\text{-}//\text{-})\ [90,55,90]$
$50)$
**translations** $\Vert-\ p\ c\ q \rightleftharpoons (p,\ c,\ q) \in oghoare$

**inductive** *oghoare ann-hoare*
**intros**
*AnnBasic*: $r \subseteq \{s.\ f\ s \in q\} \Longrightarrow \vdash (AnnBasic\ r\ f)\ q$

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

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

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

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

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

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

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

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

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

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

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

## 1.5 Soundness

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

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

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

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

### 1.5.1 Soundness of the System for Atomic Programs

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

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

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

### 1.5.2 Soundness of the System for Component Programs

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

Strong Soundness for Component Programs:

**lemma** *ann-hoare-case-analysis* [*rule-format*]: $\vdash C\ q' \longrightarrow$
$((\forall r\,f.\ C = \mathit{AnnBasic}\ r\ f \longrightarrow (\exists q.\ r \subseteq \{s.\ f\ s \in q\} \wedge q \subseteq q')) \wedge$
$(\forall c0\ c1.\ C = \mathit{AnnSeq}\ c0\ c1 \longrightarrow (\exists q.\ q \subseteq q' \wedge \vdash c0\ \mathit{pre}\ c1 \wedge \vdash c1\ q)) \wedge$

$(\forall r\ b\ c1\ c2.\ C = AnnCond1\ r\ b\ c1\ c2 \longrightarrow (\exists q.\ q \subseteq q' \wedge$
$r \cap b \subseteq pre\ c1\ \wedge \vdash c1\ q \wedge r \cap -b \subseteq pre\ c2 \wedge \vdash c2\ q)) \wedge$
$(\forall r\ b\ c.\ C = AnnCond2\ r\ b\ c \longrightarrow$
$(\exists q.\ q \subseteq q' \wedge r \cap b \subseteq pre\ c\ \wedge \vdash c\ q \wedge r \cap -b \subseteq q)) \wedge$
$(\forall r\ i\ b\ c.\ C = AnnWhile\ r\ b\ i\ c \longrightarrow$
$(\exists q.\ q \subseteq q' \wedge r \subseteq i \wedge i \cap b \subseteq pre\ c \wedge \vdash c\ i \wedge i \cap -b \subseteq q)) \wedge$
$(\forall r\ b\ c.\ C = AnnAwait\ r\ b\ c \longrightarrow (\exists q.\ q \subseteq q' \wedge \Vert- (r \cap b)\ c\ q)))$
$\langle proof \rangle$

**lemma** *Help*: $(transition \cap \{(v,v,u).\ True\}) = (transition)$
$\langle proof \rangle$

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

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

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

**lemma** *ann-hoare-sound*: $\vdash c\ q\ \Longrightarrow \models c\ q$
$\langle proof \rangle$

### 1.5.3 Soundness of the System for Parallel Programs

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

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

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

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

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

$\langle proof \rangle$

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

Strong Soundness Theorem for Parallel Programs:

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

**lemma** *Parallel-Strong-Soundness-Seq* [*rule-format* (*no-asm*)]:
$[\![\ \forall i{<}length\ Ts.\ (if\ com(Ts!i) = None\ then\ b \in post(Ts!i)$
$else\ b \in pre(the(com(Ts!i))) \wedge \vdash the(com(Ts!i))\ post(Ts!i));$
$com(Ts\ !\ i) = Some(AnnSeq\ c0\ c1);\ i{<}length\ Ts;\ interfree\ Ts\ ]\!] \Longrightarrow$
$(\forall ia{<}length\ Ts.\ (if\ com(Ts[i{:=}(Some\ c0,\ pre\ c1)]!\ ia) = None$
$then\ b \in post(Ts[i{:=}(Some\ c0,\ pre\ c1)]!\ ia)$
$else\ b \in pre(the(com(Ts[i{:=}(Some\ c0,\ pre\ c1)]!\ ia))) \wedge$
$\vdash the(com(Ts[i{:=}(Some\ c0,\ pre\ c1)]!\ ia))\ post(Ts[i{:=}(Some\ c0,\ pre\ c1)]!\ ia)))$
$\wedge\ interfree\ (Ts[i{:=}\ (Some\ c0,\ pre\ c1)])$
$\langle proof \rangle$

**lemma** *Parallel-Strong-Soundness-aux-aux* [*rule-format*]:
$(Some\ c,\ b)\ -1\rightarrow\ (co,\ t) \longrightarrow$
$(\forall Ts.\ i{<}length\ Ts \longrightarrow com(Ts\ !\ i) = Some\ c \longrightarrow$
$(\forall i{<}length\ Ts.\ (if\ com(Ts\ !\ i) = None\ then\ b{\in}post(Ts!i)$
$else\ b{\in}pre(the(com(Ts!i))) \wedge \vdash the(com(Ts!i))\ post(Ts!i))) \longrightarrow$
$interfree\ Ts \longrightarrow$
$(\forall j.\ j{<}length\ Ts \wedge i{\neq}j \longrightarrow (if\ com(Ts!j) = None\ then\ t{\in}post(Ts!j)$
$else\ t{\in}pre(the(com(Ts!j))) \wedge \vdash the(com(Ts!j))\ post(Ts!j))))\ )$
$\langle proof \rangle$

**lemma** *Parallel-Strong-Soundness-aux* [*rule-format*]:
$[\![(Ts',s)\ -P{*}\rightarrow\ (Rs',t);\ \ Ts' = (Parallel\ Ts);\ interfree\ Ts;$
$\forall i.\ i{<}length\ Ts \longrightarrow (\exists c\ q.\ (Ts\ !\ i) = (Some\ c,\ q) \wedge s{\in}(pre\ c) \wedge \vdash c\ q\ )\ ]\!] \Longrightarrow$
$\forall Rs.\ Rs' = (Parallel\ Rs) \longrightarrow (\forall j.\ j{<}length\ Rs \longrightarrow$
$(if\ com(Rs\ !\ j) = None\ then\ t{\in}post(Ts\ !\ j)$
$else\ t{\in}pre(the(com(Rs\ !\ j))) \wedge \vdash the(com(Rs\ !\ j))\ post(Ts\ !\ j))) \wedge interfree\ Rs$
$\langle proof \rangle$

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

**lemma** *oghoare-sound* [*rule-format*]: $\|{-}\ p\ c\ q \longrightarrow \|{=}\ p\ c\ q$
$\langle proof \rangle$

**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*:
$[\![\ p \subseteq (\bigcap i \in \{i.\ i < length\ Ts\}.\ pre(the(com(Ts\ !\ i))));$
$\Vert - (\bigcap i \in \{i.\ i < length\ Ts\}.\ pre(the(com(Ts\ !\ i))))$
$(Parallel\ Ts)$
$(\bigcap i \in \{i.\ i < length\ Ts\}.\ post(Ts\ !\ i));$
$(\bigcap i \in \{i.\ i < length\ Ts\}.\ post(Ts\ !\ i)) \subseteq q\ ]\!]$
$\Longrightarrow \Vert - p\ (Parallel\ Ts)\ q$
⟨*proof*⟩

**lemma** *SkipRule*: $p \subseteq q \Longrightarrow \Vert - p\ (Basic\ id)\ q$
⟨*proof*⟩

**lemma** *BasicRule*: $p \subseteq \{s.\ (f\ s) \in q\} \Longrightarrow \Vert - p\ (Basic\ f)\ q$
⟨*proof*⟩

**lemma** *SeqRule*: $[\![\ \Vert - p\ c1\ r;\ \Vert - r\ c2\ q\ ]\!] \Longrightarrow \Vert - p\ (Seq\ c1\ c2)\ q$
⟨*proof*⟩

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

**lemma** *WhileRule*: $[\![\ p \subseteq i;\ \Vert - (i \cap b)\ c\ i\ ;\ (i \cap (-b)) \subseteq q\ ]\!]$
$\Longrightarrow \Vert - p\ (While\ b\ i\ c)\ q$
⟨*proof*⟩

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

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

**lemma** *AnnskipRule*:

$r \subseteq q \Longrightarrow \vdash (AnnBasic\ r\ id)\ q$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

**lemma** *AnnAwait-assertions*:

$\llbracket$ *interfree-aux*(*None*, *r*, *Some a*); *interfree-aux*(*None*, *q*, *Some a*)$\rrbracket \Longrightarrow$
*interfree-aux*(*Some* (*AnnAwait r b c*), *q*, *Some a*)
$\langle proof \rangle$

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

**lemma** *AnnSeq-atomics*:
$\llbracket$ *interfree-aux*(*Any*, *q*, *Some a1*); *interfree-aux*(*Any*, *q*, *Some a2*)$\rrbracket \Longrightarrow$
*interfree-aux*(*Any*, *q*, *Some* (*AnnSeq a1 a2*))
$\langle proof \rangle$

**lemma** *AnnCond1-atomics*:
$\llbracket$ *interfree-aux*(*Any*, *q*, *Some a1*); *interfree-aux*(*Any*, *q*, *Some a2*)$\rrbracket \Longrightarrow$
*interfree-aux*(*Any*, *q*, *Some* (*AnnCond1 r b a1 a2*))
$\langle proof \rangle$

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

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

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

**lemma** *AnnAwait-atomics*:
$\Vdash$ $(q \cap (r \cap b))$ *a q* $\Longrightarrow$ *interfree-aux* (*None*, *q*, *Some* (*AnnAwait r b a*))
$\langle proof \rangle$

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

**lemma** *interfree-swap-Empty*: *interfree-swap* (*x*, [])
$\langle proof \rangle$

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

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

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

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

**lemma** *interfree-Map*:
($\forall i\ j.\ a \le i \wedge i < b \wedge a \le j \wedge j < b \wedge i \ne j \longrightarrow$ *interfree-aux* (*c i*, *Q i*, *c j*))
$\Longrightarrow$ *interfree* (*map* ($\lambda k.$ (*c k*, *Q k*)) [*a*..<*b*])
⟨*proof*⟩

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

**lemma** *MapAnnEmpty*: [⊢] []
⟨*proof*⟩

**lemma** *MapAnnList*: ⟦ ⊢ *c q* ; [⊢] *xs* ⟧ $\Longrightarrow$ [⊢] (*Some c*,*q*)#*xs*
⟨*proof*⟩

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

**lemma** *ParallelRule*:⟦ [⊢] *Ts* ; *interfree Ts* ⟧
$\Longrightarrow$ ‖− ($\bigcap i \in \{i.\ i <$ *length Ts*$\}$. *pre*(*the*(*com*(*Ts*!*i*))))
*Parallel Ts*
($\bigcap i \in \{i.\ i <$ *length Ts*$\}$. *post*(*Ts*!*i*))
⟨*proof*⟩

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

**lemma** *Compl-Collect*: $-$(*Collect b*) = $\{x.\ \neg(b\ x)\}$
⟨*proof*⟩
**lemma** *list-length*: *length* []=*0* $\wedge$ *length* (*x*#*xs*) = *Suc*(*length xs*)
⟨*proof*⟩
**lemma** *list-lemmas*: *length* []=*0* $\wedge$ *length* (*x*#*xs*) = *Suc*(*length xs*)
$\wedge$ (*x*#*xs*) ! *0*=*x* $\wedge$ (*x*#*xs*) ! *Suc n* = *xs* ! *n*
⟨*proof*⟩
**lemma** *le-Suc-eq-insert*: $\{i.\ i <$ *Suc n*$\}$ = *insert n* $\{i.\ i < n\}$
⟨*proof*⟩

**lemmas** *primrecdef-list = pre.simps assertions.simps atomics.simps atom-com.simps*
**lemmas** *my-simp-list = list-lemmas fst-conv snd-conv*
*not-less0 refl le-Suc-eq-insert Suc-not-Zero Zero-not-Suc 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 $A \Longrightarrow a1 \wedge a2 \wedge .. \wedge an$ returning $n$ subgoals, one for each conjunct:

⟨*ML*⟩

### Tactic for the generation of the verification conditions

The tactic basically uses two subtactics:

**HoareRuleTac** is called at the level of parallel programs, it uses the ParallelTac to solve parallel composition of programs. This verification has two parts, namely, (1) all component programs are correct and (2) they are interference free. *HoareRuleTac* is also called at the level of atomic regions, i.e. ⟨ ⟩ 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* $\subseteq \ldots$ 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* :: $'b \Rightarrow ('a \Rightarrow 'b)$ ((«-») [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*}. ⇀ *Collect* «*b*»

⟨*ML*⟩

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

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

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

**translations**
´ *x* := *a* ⇀ *Basic* «´ (*-update-name x a*)»
*r* ´ *x* := *a* ⇀ *AnnBasic r* «´ (*-update-name x a*)»

**syntax**
*-AnnSkip* :: $'a$ *assn* ⇒ $'a$ *ann-com* (-//SKIP [90] 63)
*-AnnSeq* :: $'a$ *ann-com* ⇒ $'a$ *ann-com* ⇒ $'a$ *ann-com* (-;;/ - [60,61] 60)

*-AnnCond1* :: $'a$ *assn* ⇒ $'a$ *bexp* ⇒ $'a$ *ann-com* ⇒ $'a$ *ann-com* ⇒ $'a$ *ann-com*
(- //IF - /THEN - /ELSE - /FI [90,0,0,0] 61)
*-AnnCond2* :: $'a$ *assn* ⇒ $'a$ *bexp* ⇒ $'a$ *ann-com* ⇒ $'a$ *ann-com*
(- //IF - /THEN - /FI [90,0,0] 61)
*-AnnWhile* :: $'a$ *assn* ⇒ $'a$ *bexp* ⇒ $'a$ *assn* ⇒ $'a$ *ann-com* ⇒ $'a$ *ann-com*

(- //WHILE - /INV - //DO -//OD [90,0,0,0] 61)
-AnnAwait :: 'a assn ⇒ 'a bexp ⇒ 'a com ⇒ 'a ann-com
(- //AWAIT - /THEN /- /END [90,0,0] 61)
-AnnAtom :: 'a assn ⇒ 'a com ⇒ 'a ann-com (-//⟨-⟩ [90,0] 61)
-AnnWait :: 'a assn ⇒ 'a bexp ⇒ 'a ann-com (-//WAIT - END [90,0] 61)

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

**translations**

SKIP ⇌ Basic id
c-1,, c-2 ⇌ Seq c-1 c-2

IF b THEN c1 ELSE c2 FI ⇀ Cond .{b}. c1 c2
IF b THEN c FI ⇌ IF b THEN c ELSE SKIP FI
WHILE b INV i DO c OD ⇀ While .{b}. i c
WHILE b DO c OD ⇌ WHILE b INV arbitrary DO c OD

r SKIP ⇌ AnnBasic r id
c-1;; c-2 ⇌ AnnSeq c-1 c-2
r IF b THEN c1 ELSE c2 FI ⇀ AnnCond1 r .{b}. c1 c2
r IF b THEN c FI ⇀ AnnCond2 r .{b}. c
r WHILE b INV i DO c OD ⇀ AnnWhile r .{b}. i c
r AWAIT b THEN c END ⇀ AnnAwait r .{b}. c
r ⟨c⟩ ⇌ r AWAIT True THEN c END
r WAIT b END ⇌ r AWAIT b THEN SKIP END

**nonterminals**

prgs

**syntax**

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

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

**translations**

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

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

⟨*ML*⟩

**end**

## 1.8 Examples

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

### 1.8.1 Mutual Exclusion

#### Peterson's Algorithm I

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

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

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

### Peterson's Algorithm II: A Busy Wait Solution

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

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

**lemma** *Busy-wait-mutex*:
```
 ‖−  .{True}.
  ´flag1:=False,, ´flag2:=False,,
  COBEGIN .{¬´flag1}.
        WHILE True
        INV .{¬´flag1}.
        DO .{¬´flag1}. ⟨ ´flag1:=True,,´after1:=False ⟩;;
           .{´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*:

$\|- .\{i{\neq}j\}.$
$\acute{}out{:=}True$ ,,
$COBEGIN\ .\{i{\neq}j\}.$
$\quad WHILE\ True\ INV\ .\{i{\neq}j\}.$
$\quad DO\ .\{i{\neq}j\}.\ AWAIT\ \acute{}out\ THEN\ \ \acute{}out{:=}False,,\ \acute{}who{:=}i\ END;;$
$\qquad .\{\neg\acute{}out \wedge \acute{}who{=}i \wedge i{\neq}j\}.\ \acute{}out{:=}True\ OD$
$\quad .\{False\}.$
$\|$
$\quad .\{i{\neq}j\}.$
$\quad WHILE\ True\ INV\ .\{i{\neq}j\}.$
$\quad DO\ .\{i{\neq}j\}.\ AWAIT\ \acute{}out\ THEN\ \ \acute{}out{:=}False,,\acute{}who{:=}j\ END;;$
$\qquad .\{\neg\acute{}out \wedge \acute{}who{=}j \wedge i{\neq}j\}.\ \acute{}out{:=}True\ OD$
$\quad .\{False\}.$
$COEND$
$.\{False\}.$
$\langle proof\rangle$

### Peterson's Algorithm III: Parameterized version:

**lemma** *Semaphores-parameterized-mutex*:

$0{<}n \Longrightarrow \|- .\{True\}.$
$\acute{}out{:=}True$ ,,
$COBEGIN$
$SCHEME\ [0{\le}\ i{<}\ n]$
$\quad .\{True\}.$
$\quad WHILE\ True\ INV\ .\{True\}.$
$\quad DO\ .\{True\}.\ AWAIT\ \acute{}out\ THEN\ \ \acute{}out{:=}False,,\ \acute{}who{:=}i\ END;;$
$\qquad .\{\neg\acute{}out \wedge \acute{}who{=}i\}.\ \acute{}out{:=}True\ OD$
$\quad .\{False\}.$
$COEND$
$.\{False\}.$
$\langle proof\rangle$

### The Ticket Algorithm

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

**lemma** *Ticket-mutex*:

$[\![\ 0{<}n;\ I{=}\ll n{=}length\ \acute{}turn \wedge 0{<}\acute{}nextv \wedge (\forall k\ l.\ k{<}n \wedge l{<}n \wedge k{\neq}l$
$\quad \longrightarrow \acute{}turn!k < \acute{}num \wedge (\acute{}turn!k = 0 \vee \acute{}turn!k{\neq}\acute{}turn!l))\gg\ ]\!]$
$\Longrightarrow \|- .\{n{=}length\ \acute{}turn\}.$
$\acute{}index{:=}\ 0,,$
$WHILE\ \acute{}index < n\ INV\ .\{n{=}length\ \acute{}turn \wedge (\forall i{<}\acute{}index.\ \acute{}turn!i{=}0)\}.$

```
  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}.
```

$\parallel$
.{´*I2*}.
*WHILE* $\neg$´*found*
*INV* .{´*I2*}.
*DO* .{´$y \leq a+1$ $\wedge$ (´*found* $\longrightarrow$ $a <$´$x$ $\wedge$ $f$(´$x$)$=0$) $\wedge$ (´$y \leq a$ $\longrightarrow$ $f$(´$y$)$\neq 0$)}.
*WAIT* ´*turn*$=2$ *END*;;
.{´$y \leq a+1$ $\wedge$ (´*found* $\longrightarrow$ $a <$´$x$ $\wedge$ $f$(´$x$)$=0$) $\wedge$ (´$y \leq a$ $\longrightarrow$ $f$(´$y$)$\neq 0$)}.
´*turn*$:=1$;;
.{´$y \leq a+1$ $\wedge$ (´*found* $\longrightarrow$ $a <$´$x$ $\wedge$ $f$(´$x$)$=0$) $\wedge$ (´$y \leq a$ $\longrightarrow$ $f$(´$y$)$\neq 0$)}.
$\langle$ ´$y$:=(´$y$ $-$ $1$),,
*IF* $f$(´$y$)$=0$ *THEN* ´*found*:=*True* *ELSE* *SKIP* *FI*$\rangle$
*OD*;;
.{´*I2* $\wedge$ ´*found*}.
´*turn*$:=1$
.{´*I2* $\wedge$ ´*found*}.
*COEND*
.{$f$(´$x$)$=0$ $\vee$ $f$(´$y$)$=0$}.
$\langle$*proof*$\rangle$

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

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

### 1.8.3 Producer/Consumer

#### Previous lemmas

**lemma** *nat-lemma2*: ⟦ $b = m*(n::nat) + t$; $a = s*n + u$; $t=u$; $b-a < n$ ⟧ $\Longrightarrow$ $m \le s$
⟨*proof*⟩

**lemma** *mod-lemma*: ⟦ $(c::nat) \le a$; $a < b$; $b - c < n$ ⟧ $\Longrightarrow$ $b$ *mod* $n \neq a$ *mod* $n$
⟨*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*:
  ⟦*INIT*= ≪0<length a ∧ 0<length ´buffer ∧ length ´b=length a≫ ;
    *I*= ≪(∀ k<´ins. ´outs≤k ⟶ (a ! k) = ´buffer ! (k mod (length ´buffer))) ∧
          ´outs≤´ins ∧ ´ins−´outs≤length ´buffer≫ ;
    *I1*= ≪´I ∧ ´li≤length a≫ ;
    *p1*= ≪´I1 ∧ ´li=´ins≫ ;
    *I2* = ≪´I ∧ (∀ k<´lj. (a ! k)=(´b ! k)) ∧ ´lj≤length a≫ ;
    *p2* = ≪´I2 ∧ ´lj=´outs≫ ⟧ ⟹
  ‖− .{´INIT}.
 ´ins:=0,, ´outs:=0,, ´li:=0,, ´lj:=0,,
 COBEGIN .{´p1 ∧ ´INIT}.
  WHILE ´li <length a
   INV .{´p1 ∧ ´INIT}.
  DO .{´p1 ∧ ´INIT ∧ ´li<length a}.
    ´vx:= (a ! ´li);;
   .{´p1 ∧ ´INIT ∧ ´li<length a ∧ ´vx=(a ! ´li)}.
     WAIT ´ins−´outs < length ´buffer END;;
   .{´p1 ∧ ´INIT ∧ ´li<length a ∧ ´vx=(a ! ´li)
     ∧ ´ins−´outs < length ´buffer}.
    ´buffer:=(list-update ´buffer (´ins mod (length ´buffer)) ´vx);;
   .{´p1 ∧ ´INIT ∧ ´li<length a
     ∧ (a ! ´li)=(´buffer ! (´ins mod (length ´buffer)))
     ∧ ´ins−´outs <length ´buffer}.
    ´ins:=´ins+1;;
   .{´I1 ∧ ´INIT ∧ (´li+1)=´ins ∧ ´li<length a}.
    ´li:=´li+1

$OD$
$.\{ ´p1 \wedge ´INIT \wedge ´li{=}length\ a\}.$
$\|$
$.\{ ´p2 \wedge ´INIT\}.$
$WHILE\ ´lj < length\ a$
$INV\ .\{ ´p2 \wedge ´INIT\}.$
$DO\ .\{ ´p2 \wedge ´lj{<}length\ a \wedge ´INIT\}.$
$WAIT\ ´outs{<}´ins\ END;;$
$.\{ ´p2 \wedge ´lj{<}length\ a \wedge ´outs{<}´ins \wedge ´INIT\}.$
$´vy{:=}(´buffer\ !\ (´outs\ mod\ (length\ ´buffer)));;$
$.\{ ´p2 \wedge ´lj{<}length\ a \wedge ´outs{<}´ins \wedge ´vy{=}(a\ !\ ´lj) \wedge ´INIT\}.$
$´outs{:=}´outs{+}1;;$
$.\{ ´I2 \wedge (´lj{+}1){=}´outs \wedge ´lj{<}length\ a \wedge ´vy{=}(a\ !\ ´lj) \wedge ´INIT\}.$
$´b{:=}(list\text{-}update\ ´b\ ´lj\ ´vy);;$
$.\{ ´I2 \wedge (´lj{+}1){=}´outs \wedge ´lj{<}length\ a \wedge (a\ !\ ´lj){=}(´b\ !\ ´lj) \wedge ´INIT\}.$
$´lj{:=}´lj{+}1$
$OD$
$.\{ ´p2 \wedge ´lj{=}length\ a \wedge ´INIT\}.$
$COEND$
$.\{ \forall k{<}length\ a.\ (a\ !\ k){=}(´b\ !\ k)\}.$
$\langle proof\rangle$

### 1.8.4 Parameterized Examples

#### Set Elements of an Array to Zero

**record** *Example1* =
$a :: nat \Rightarrow nat$

**lemma** *Example1*:
$\|-\ .\{True\}.$
$COBEGIN\ SCHEME\ [0{\le}i{<}n]\ .\{True\}.\ ´a{:=}´a\ (i{:=}0)\ .\{´a\ i{=}0\}.\ COEND$
$.\{\forall i < n.\ ´a\ i = 0\}.$
$\langle proof\rangle$

Same example with lists as auxiliary variables.

**record** *Example1-list* =
$A :: nat\ list$
**lemma** *Example1-list*:
$\|-\ .\{n < length\ ´A\}.$
$COBEGIN$
$SCHEME\ [0{\le}i{<}n]\ .\{n < length\ ´A\}.\ ´A{:=}´A[i{:=}0]\ .\{´A!i{=}0\}.$
$COEND$
$.\{\forall i < n.\ ´A!i = 0\}.$
$\langle proof\rangle$

#### Increment a Variable in Parallel

First some lemmas about summation properties.

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

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

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

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

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

**end**

# Chapter 2

# Case Study: Single and Multi-Mutator Garbage Collection Algorithms

## 2.1 Formalization of the Memory

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

**datatype** $node = Black \mid White$

**types**
  $nodes = node\ list$
  $edge\ \ = nat \times nat$
  $edges = edge\ list$

**consts** $Roots :: nat\ set$

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

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

  $BtoW :: (edge \times nodes) \Rightarrow bool$
  $BtoW \equiv (\lambda(e,M).\ (M!fst\ e)=Black \wedge (M!snd\ e)\neq Black)$

  $Blacks :: nodes \Rightarrow nat\ set$
  $Blacks\ M \equiv \{i.\ i<length\ M \wedge M!i=Black\}$

  $Reach :: edges \Rightarrow nat\ set$
  $Reach\ E \equiv \{x.\ (\exists path.\ 1<length\ path \wedge path!(length\ path - 1)\in Roots \wedge x=path!0$

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

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

### 2.1.1 Proofs about Graphs

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

#### Graph 1

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

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

#### Graph 2

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

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

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

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

#### Graph 3

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

### Graph 4

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

### Graph 5

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

### Other lemmas about graphs

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

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

**lemma** *Graph8*:
$[\![Proper\text{-}Roots(M)]\!] \Longrightarrow Proper\text{-}Roots(M[T:=a])$
⟨*proof*⟩

Some specific lemmata for the verification of garbage collection algorithms.

**lemma** *Graph9*: $j<length\ M \Longrightarrow Blacks\ M \subseteq Blacks\ (M[j := Black])$
⟨*proof*⟩

**lemma** *Graph10* [*rule-format* (*no-asm*)]: $\forall\, i.\ M!i=a \longrightarrow M[i:=a]=M$
⟨*proof*⟩

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

**lemma** *Graph12*: $[\![a \subseteq Blacks\ M;j<length\ M]\!] \Longrightarrow a \subseteq Blacks\ (M[j := Black])$
⟨*proof*⟩

**lemma** *Graph13*: $[\![a \subset Blacks\ M;j<length\ M]\!] \Longrightarrow a \subset Blacks\ (M[j := Black])$
⟨*proof*⟩

**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** $R :: nat \quad T :: nat$

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**
  $\mathit{Mut\text{-}init} :: \mathit{gar\text{-}coll\text{-}state} \Rightarrow \mathit{bool}$
  $\mathit{Mut\text{-}init} \equiv \ll T \in \mathit{Reach}\ {}^\acute{}E \wedge R < \mathit{length}\ {}^\acute{}E \wedge T < \mathit{length}\ {}^\acute{}M \gg$

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

**constdefs**
  $\mathit{Redirect\text{-}Edge} :: \mathit{gar\text{-}coll\text{-}state\ ann\text{-}com}$
  $\mathit{Redirect\text{-}Edge} \equiv .\{{}^\acute{}\mathit{Mut\text{-}init} \wedge {}^\acute{}z\}.\ \langle {}^\acute{}E{:=}{}^\acute{}E[R{:=}(\mathit{fst}({}^\acute{}E!R),\ T)],,\ {}^\acute{}z{:=}\ (\neg {}^\acute{}z)\rangle$

  $\mathit{Color\text{-}Target} :: \mathit{gar\text{-}coll\text{-}state\ ann\text{-}com}$
  $\mathit{Color\text{-}Target} \equiv .\{{}^\acute{}\mathit{Mut\text{-}init} \wedge \neg {}^\acute{}z\}.\ \langle {}^\acute{}M{:=}{}^\acute{}M[T{:=}\mathit{Black}],,\ {}^\acute{}z{:=}\ (\neg {}^\acute{}z)\rangle$

  $\mathit{Mutator} :: \mathit{gar\text{-}coll\text{-}state\ ann\text{-}com}$
  $\mathit{Mutator} \equiv$
  $.\{{}^\acute{}\mathit{Mut\text{-}init} \wedge {}^\acute{}z\}.$

*WHILE True INV* .{´*Mut-init* ∧ ´*z*}.
*DO Redirect-Edge* ;; *Color-Target OD*

### Correctness of the mutator

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

**lemma** *Redirect-Edge*:
⊢ *Redirect-Edge pre*(*Color-Target*)
⟨*proof*⟩

**lemma** *Color-Target*:
⊢ *Color-Target* .{´*Mut-init* ∧ ´*z*}.
⟨*proof*⟩

**lemma** *Mutator*:
⊢ *Mutator* .{*False*}.
⟨*proof*⟩

## 2.2.2 The Collector

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

**consts** *M-init* :: *nodes*

**constdefs**
*Proper-M-init* :: *gar-coll-state* ⇒ *bool*
*Proper-M-init* ≡ « *Blacks M-init*=*Roots* ∧ *length M-init*=*length* ´*M* »

*Proper* :: *gar-coll-state* ⇒ *bool*
*Proper* ≡ « *Proper-Roots* ´*M* ∧ *Proper-Edges*(´*M*, ´*E*) ∧ ´*Proper-M-init* »

*Safe* :: *gar-coll-state* ⇒ *bool*
*Safe* ≡ « *Reach* ´*E* ⊆ *Blacks* ´*M* »

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

### Blackening the roots

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

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

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

### Propagating black

**constdefs**
*PBInv* :: *gar-coll-state* ⇒ *nat* ⇒ *bool*
*PBInv* ≡ ≪ λ*ind*. ´*obc* < *Blacks* ´*M* ∨ (∀ *i* <*ind*. ¬*BtoW* (´*E*!*i*, ´*M*) ∨
(¬´*z* ∧ *i*=*R* ∧ (*snd*(´*E*!*R*)) = *T* ∧ (∃ *r*. *ind* ≤ *r* ∧ *r* < *length* ´*E* ∧ *BtoW*(´*E*!*r*,´*M*))))≫

**constdefs**
*Propagate-Black-aux* :: *gar-coll-state ann-com*
*Propagate-Black-aux* ≡
.{´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*}.
´*ind*:=*0*;;
.{´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M* ∧ ´*ind*=*0*}.

*WHILE* ´*ind*<*length* ´*E*
*INV* .{´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ ´*PBInv* ´*ind* ∧ ´*ind*≤*length* ´*E*}.
*DO* .{´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ ´*PBInv* ´*ind* ∧ ´*ind*<*length* ´*E*}.
*IF* ´*M*!(*fst* (´*E*!´*ind*)) = *Black THEN*
.{´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ ´*PBInv* ´*ind* ∧ ´*ind*<*length* ´*E* ∧ ´*M*!*fst*(´*E*!´*ind*)=*Black*}.
´*M*:=´*M*[*snd*(´*E*!´*ind*):=*Black*];;
.{´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ ´*PBInv* (´*ind* + *1*) ∧ ´*ind*<*length* ´*E*}.
´*ind*:=´*ind*+*1*
*FI*
*OD*

**lemma** *Propagate-Black-aux*:
⊢ *Propagate-Black-aux*
.{´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ ( ´*obc* < *Blacks* ´*M* ∨ ´*Safe*)}.
⟨*proof*⟩

### Refining propagating black

**constdefs**
*Auxk* :: *gar-coll-state* ⇒ *bool*

```
Auxk ≡ «´k<length ´M ∧ (´M!´k≠Black ∨ ¬BtoW(´E!´ind, ´M) ∨
        ´obc<Blacks ´M ∨ (¬´z ∧ ´ind=R ∧ snd(´E!R)=T
        ∧ (∃ r. ´ind<r ∧ r<length ´E ∧ BtoW(´E!r, ´M))))»
```

**constdefs**

```
Propagate-Black :: gar-coll-state ann-com
Propagate-Black ≡
.{´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M}.
´ind:=0;;
.{´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M ∧ ´ind=0}.
WHILE ´ind<length ´E
 INV .{´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
      ∧ ´PBInv ´ind ∧ ´ind≤length ´E}.
DO .{´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
     ∧ ´PBInv ´ind ∧ ´ind<length ´E}.
 IF (´M!(fst (´E!´ind)))=Black THEN
  .{´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
    ∧ ´PBInv ´ind ∧ ´ind<length ´E ∧ (´M!fst(´E!´ind))=Black}.
   ´k:=(snd(´E!´ind));;
  .{´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
    ∧ ´PBInv ´ind ∧ ´ind<length ´E ∧ (´M!fst(´E!´ind))=Black
    ∧ ´Auxk}.
   ⟨´M:=´M[´k:=Black],, ´ind:=´ind+1⟩
 ELSE .{´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
      ∧ ´PBInv ´ind ∧ ´ind<length ´E}.
      ⟨IF (´M!(fst (´E!´ind)))≠Black THEN ´ind:=´ind+1 FI⟩
 FI
OD
```

**lemma** *Propagate-Black*:

```
 ⊢ Propagate-Black
 .{´Proper ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
   ∧ ( ´obc < Blacks ´M ∨ ´Safe)}.
⟨proof⟩
```

### Counting black nodes

**constdefs**

```
CountInv :: gar-coll-state ⇒ nat ⇒ bool
CountInv ≡ « λind. {i. i<ind ∧ ´Ma!i=Black}⊆´bc »
```

**constdefs**

```
Count :: gar-coll-state ann-com
Count ≡
.{´Proper ∧ Roots⊆Blacks ´M
  ∧ ´obc⊆Blacks ´Ma ∧ Blacks ´Ma⊆Blacks ´M ∧ ´bc⊆Blacks ´M
  ∧ length ´Ma=length ´M ∧ (´obc < Blacks ´Ma ∨ ´Safe) ∧ ´bc={}}.
´ind:=0;;
.{´Proper ∧ Roots⊆Blacks ´M
```

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

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

### Appending garbage nodes to the free list

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

**axioms**
*Append-to-free0*: *length* (*Append-to-free* (*i*, *e*)) = *length e*
*Append-to-free1*: *Proper-Edges* (*m*, *e*)
  $\Longrightarrow$ *Proper-Edges* (*m*, *Append-to-free*(*i*, *e*))
*Append-to-free2*: *i* $\notin$ *Reach e*
  $\Longrightarrow$ *n* $\in$ *Reach* (*Append-to-free*(*i*, *e*)) = ( *n* = *i* $\vee$ *n* $\in$ *Reach e*)

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

**constdefs**
*Append* :: *gar-coll-state ann-com*
*Append* ≡
$.\{{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime Safe\}.$
${}^\prime ind{:=}0;;$
$.\{{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime Safe \wedge {}^\prime ind{=}0\}.$
*WHILE* $ {}^\prime ind{<}length\ {}^\prime M$
*INV* $.\{{}^\prime Proper \wedge {}^\prime AppendInv\ {}^\prime ind \wedge {}^\prime ind{\leq}length\ {}^\prime M\}.$
*DO* $.\{{}^\prime Proper \wedge {}^\prime AppendInv\ {}^\prime ind \wedge {}^\prime ind{<}length\ {}^\prime M\}.$
*IF* ${}^\prime M!{}^\prime ind{=}Black$ *THEN*
$.\{{}^\prime Proper \wedge {}^\prime AppendInv\ {}^\prime ind \wedge {}^\prime ind{<}length\ {}^\prime M \wedge {}^\prime M!{}^\prime ind{=}Black\}.$
${}^\prime M{:=}{}^\prime M[{}^\prime ind{:=}White]$
*ELSE* $.\{{}^\prime Proper \wedge {}^\prime AppendInv\ {}^\prime ind \wedge {}^\prime ind{<}length\ {}^\prime M \wedge {}^\prime ind{\notin}Reach\ {}^\prime E\}.$
${}^\prime E{:=}Append\text{-}to\text{-}free({}^\prime ind,{}^\prime E)$
*FI*;;
$.\{{}^\prime Proper \wedge {}^\prime AppendInv\ ({}^\prime ind{+}1) \wedge {}^\prime ind{<}length\ {}^\prime M\}.$
${}^\prime ind{:=}{}^\prime ind{+}1$
*OD*

**lemma** *Append*:
$\vdash$ *Append* $.\{{}^\prime Proper\}.$
⟨*proof*⟩

### Correctness of the Collector

**constdefs**
*Collector* :: *gar-coll-state ann-com*
*Collector* ≡
$.\{{}^\prime Proper\}.$
*WHILE True INV* $.\{{}^\prime Proper\}.$
*DO*
*Blacken-Roots*;;
$.\{{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M\}.$
${}^\prime obc{:=}\{\};;$
$.\{{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{=}\{\}\}.$
${}^\prime bc{:=}Roots;;$
$.\{{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{=}\{\} \wedge {}^\prime bc{=}Roots\}.$
${}^\prime Ma{:=}M\text{-}init;;$
$.\{{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{=}\{\} \wedge {}^\prime bc{=}Roots \wedge {}^\prime Ma{=}M\text{-}init\}.$
*WHILE* ${}^\prime obc{\neq}{}^\prime bc$
*INV* $.\{{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M$
$\wedge\ {}^\prime obc{\subseteq}Blacks\ {}^\prime Ma \wedge Blacks\ {}^\prime Ma{\subseteq}{}^\prime bc \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M$
$\wedge\ length\ {}^\prime Ma{=}length\ {}^\prime M \wedge ({}^\prime obc < Blacks\ {}^\prime Ma \vee {}^\prime Safe)\}.$
*DO* $.\{{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M\}.$
${}^\prime obc{:=}{}^\prime bc;;$
*Propagate-Black*;;
$.\{{}^\prime Proper \wedge Roots{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime obc{\subseteq}Blacks\ {}^\prime M \wedge {}^\prime bc{\subseteq}Blacks\ {}^\prime M$
$\wedge\ ({}^\prime obc < Blacks\ {}^\prime M \vee {}^\prime Safe)\}.$
${}^\prime Ma{:=}{}^\prime M;;$

.{´*Proper* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*⊆*Blacks* ´*Ma*
∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M* ∧ *length* ´*Ma*=*length* ´*M*
∧ ( ´*obc* < *Blacks* ´*Ma* ∨ ´*Safe*)}.
´*bc*:={};;
*Count*
*OD*;;
*Append*
*OD*

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

### 2.2.3 Interference Freedom

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### Interference freedom Collector-Mutator

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

$\langle proof \rangle$

**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*:
$\|-$ .{´*Proper* ∧ ´*Mut-init* ∧ ´*z*}.
*COBEGIN*
*Collector*
.{*False*}.
$\|$
*Mutator*
.{*False*}.
*COEND*
.{*False*}.
$\langle proof \rangle$

**end**

## 2.3 The Multi-Mutator Case

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

The full theory takes aprox. 18 minutes.

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

Declaration of variables:

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

### 2.3.1 The Mutators

**constdefs**

*Mul-mut-init* :: *mul-gar-coll-state* ⇒ *nat* ⇒ *bool*
*Mul-mut-init* ≡ ≪ $\lambda n.\ n{=}length\ {}^\prime Muts \wedge (\forall\, i{<}n.\ R\ ({}^\prime Muts!i){<}length\ {}^\prime E$
$\wedge\ T\ ({}^\prime Muts!i){<}length\ {}^\prime M)$ ≫

*Mul-Redirect-Edge* :: *nat* ⇒ *nat* ⇒ *mul-gar-coll-state ann-com*
*Mul-Redirect-Edge j n* ≡
.{´*Mul-mut-init n* ∧ *Z* (´*Muts*!*j*)}.
⟨*IF T*(´*Muts*!*j*) ∈ *Reach* ´*E THEN*
´*E*:= ´*E*[*R* (´*Muts*!*j*):= (*fst* (´*E*!*R*(´*Muts*!*j*)), *T* (´*Muts*!*j*))] *FI*,,
´*Muts*:= ´*Muts*[*j*:= (´*Muts*!*j*) (|*Z*:=*False*|)]⟩

*Mul-Color-Target* :: *nat* ⇒ *nat* ⇒ *mul-gar-coll-state ann-com*
*Mul-Color-Target j n* ≡
.{´*Mul-mut-init n* ∧ ¬ *Z* (´*Muts*!*j*)}.
⟨´*M*:=´*M*[*T* (´*Muts*!*j*):=*Black*],, ´*Muts*:=´*Muts*[*j*:= (´*Muts*!*j*) (|*Z*:=*True*|)]⟩

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

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

#### Correctness of the proof outline of one mutator

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

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

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

#### Interference freedom between mutators

**lemma** *Mul-interfree-Redirect-Edge-Redirect-Edge*:
$[\![0 \le i;\ i{<}n;\ 0 \le j;\ j{<}n;\ i \ne j]\!] \Longrightarrow$

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

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

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

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

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

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

### Modular Parameterized Mutators

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

## 2.3.2 The Collector

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

**consts** *M-init* :: *nodes*

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

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

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

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

## Blackening Roots

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

**lemma** $Mul\text{-}Blacken\text{-}Roots$:
$\vdash Mul\text{-}Blacken\text{-}Roots\ n$
$.\{´Mul\text{-}Proper\ n \wedge Roots \subseteq Blacks\ ´M\}.$
$\langle proof\rangle$

## Propagating Black

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

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

**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-Proper n ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
 ∧ (´Safe ∨ ´l≤´Queue ∨ ´obc⊂Blacks ´M)}.
´ind:=0;;
.{´Mul-Proper n ∧ Roots⊆Blacks ´M
  ∧ ´obc⊆Blacks ´M ∧ Blacks ´M⊆Blacks ´M ∧ ´bc⊆Blacks ´M
  ∧ (´Safe ∨ ´l≤´Queue ∨ ´obc⊂Blacks ´M) ∧ ´ind=0}.
WHILE ´ind<length ´E
 INV .{´Mul-Proper n ∧ Roots⊆Blacks ´M
      ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
      ∧ ´Mul-PBInv ∧ ´ind≤length ´E}.
DO .{´Mul-Proper n ∧ Roots⊆Blacks ´M
    ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
    ∧ ´Mul-PBInv ∧ ´ind<length ´E}.
  IF ´M!(fst (´E!´ind))=Black THEN
  .{´Mul-Proper n ∧ Roots⊆Blacks ´M
    ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
    ∧ ´Mul-PBInv ∧ (´M!fst(´E!´ind))=Black ∧ ´ind<length ´E}.
   ´k:=snd(´E!´ind);;
  .{´Mul-Proper n ∧ Roots⊆Blacks ´M
    ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
    ∧ (´Safe ∨ ´obc⊂Blacks ´M ∨ ´l<´Queue ∨ (∀ i<´ind. ¬BtoW(´E!i,´M))
       ∧ ´l≤´Queue ∧ ´Mul-Auxk ) ∧ ´k<length ´M ∧ ´M!fst(´E!´ind)=Black
    ∧ ´ind<length ´E}.
  ⟨´M:=´M[´k:=Black],,´ind:=´ind+1⟩
  ELSE .{´Mul-Proper n ∧ Roots⊆Blacks ´M
       ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
       ∧ ´Mul-PBInv ∧ ´ind<length ´E}.
       ⟨IF ´M!(fst (´E!´ind))≠Black THEN ´ind:=´ind+1 FI⟩ FI
OD
```

**lemma** *Mul-Propagate-Black*:

```
 ⊢ Mul-Propagate-Black n
  .{´Mul-Proper n ∧ Roots⊆Blacks ´M ∧ ´obc⊆Blacks ´M ∧ ´bc⊆Blacks ´M
     ∧ (´Safe ∨ ´obc⊂Blacks ´M ∨ ´l<´Queue ∧ (´l≤´Queue ∨ ´obc⊂Blacks
´M))}.
⟨proof⟩
```

### Counting Black Nodes

**constdefs**

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

 Mul-Count :: nat ⇒ mul-gar-coll-state ann-com
 Mul-Count n ≡
 .{´Mul-Proper n ∧ Roots⊆Blacks ´M
   ∧ ´obc⊆Blacks ´Ma ∧ Blacks ´Ma⊆Blacks ´M ∧ ´bc⊆Blacks ´M
   ∧ length ´Ma=length ´M
   ∧ (´Safe ∨ ´obc⊂Blacks ´Ma ∨ ´l<´q ∧ (´q≤´Queue ∨ ´obc⊂Blacks ´M) )
```

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

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

### Appending garbage nodes to the free list

**consts** *Append-to-free* :: *nat* × *edges* ⇒ *edges*

**axioms**

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

**constdefs**

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

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

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

**lemma** *Mul-Append*:
$\vdash$ *Mul-Append n*
.{´*Mul-Proper n*}.
$\langle$*proof*$\rangle$

## Collector

**constdefs**

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

.{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*={}}.
´*bc*:=*Roots*;;
.{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*={} ∧ ´*bc*=*Roots*}.
´*l*:=*0*;;
.{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*obc*={} ∧ ´*bc*=*Roots* ∧ ´*l*=*0*}.
*WHILE* ´*l*<*n*+*1*
*INV* .{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M* ∧
(´*Safe* ∨ (´*l*≤´*Queue* ∨ ´*bc*⊂*Blacks* ´*M*) ∧ ´*l*<*n*+*1*)}.
*DO* .{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ (´*Safe* ∨ ´*l*≤´*Queue* ∨ ´*bc*⊂*Blacks* ´*M*)}.
´*obc*:=´*bc*;;
*Mul-Propagate-Black n*;;
.{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*M* ∨ ´*l*<´*Queue*
∧ (´*l*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*))}.
´*bc*:={};;
.{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*M* ∨ ´*l*<´*Queue*
∧ (´*l*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*)) ∧ ´*bc*={}}.
⟨ ´*Ma*:=´*M*,, ´*q*:=´*Queue* ⟩;;
*Mul-Count n*;;
.{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*Ma* ∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ *length* ´*Ma*=*length* ´*M* ∧ *Blacks* ´*Ma*⊆´*bc*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*))
∧ ´*q*<*n*+*1*}.
*IF* ´*obc*=´*bc* *THEN*
.{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*Ma* ∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ *length* ´*Ma*=*length* ´*M* ∧ *Blacks* ´*Ma*⊆´*bc*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks* ´*M*))
∧ ´*q*<*n*+*1* ∧ ´*obc*=´*bc*}.
´*l*:=´*l*+*1*
*ELSE* .{´*Mul-Proper n* ∧ *Roots*⊆*Blacks* ´*M*
∧ ´*obc*⊆*Blacks* ´*Ma* ∧ *Blacks* ´*Ma*⊆*Blacks* ´*M* ∧ ´*bc*⊆*Blacks* ´*M*
∧ *length* ´*Ma*=*length* ´*M* ∧ *Blacks* ´*Ma*⊆´*bc*
∧ (´*Safe* ∨ ´*obc*⊂*Blacks* ´*Ma* ∨ ´*l*<´*q* ∧ (´*q*≤´*Queue* ∨ ´*obc*⊂*Blacks*
´*M*))
∧ ´*q*<*n*+*1* ∧ ´*obc*≠´*bc*}.
´*l*:=*0* *FI*
*OD*;;
*Mul-Append n*
*OD*

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

**lemma** *Mul-Collector*:
$\vdash$ *Mul-Collector n*
.{*False*}.
⟨*proof*⟩

### 2.3.3 Interference Freedom

**lemma** *le-length-filter-update*[*rule-format*]:
$\forall i.\ (\neg P\ (list!i) \lor P\ j) \land i{<}length\ list$
$\longrightarrow length(filter\ P\ list) \leq length(filter\ P\ (list[i{:=}j]))$
⟨*proof*⟩

**lemma** *less-length-filter-update* [*rule-format*]:
$\forall i.\ P\ j \land \neg(P\ (list!i)) \land i{<}length\ list$
$\longrightarrow length(filter\ P\ list) < length(filter\ P\ (list[i{:=}j]))$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### Interference freedom Collector-Mutator

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

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

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

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

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

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

### Interference freedom Mutator-Collector

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

### The Multi-Mutator Garbage Collection Algorithm

The total number of verification conditions is 328

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

**end**

# Chapter 3

# The Rely-Guarantee Method

## 3.1 Abstract Syntax

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

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

**types**
$'a\ bexp = 'a\ set$

**datatype** $'a\ com =$
$\quad Basic\ 'a \Rightarrow 'a$
$\mid Seq\ 'a\ com\ 'a\ com$
$\mid Cond\ 'a\ bexp\ 'a\ com\ 'a\ com$
$\mid While\ 'a\ bexp\ 'a\ com$
$\mid Await\ 'a\ bexp\ 'a\ com$

**types** $'a\ par\text{-}com = (('a\ com)\ option)\ list$

**end**

## 3.2 Operational Semantics

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

### 3.2.1 Semantics of Component Programs

#### Environment transitions

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

**consts** *etran* $::\ ('a\ conf \times 'a\ conf)\ set$
**syntax** *-etran* $::\ 'a\ conf \Rightarrow 'a\ conf \Rightarrow bool\ \ (\text{- } -e\rightarrow \text{ - } [81,81]\ 80)$

**translations** $P\ -e\rightarrow\ Q\ \rightleftharpoons (P,Q) \in etran$
**inductive** *etran*
**intros**
*Env*: $(P,\ s)\ -e\rightarrow\ (P,\ t)$

### Component transitions

**consts** *ctran* :: $('a\ conf \times 'a\ conf)\ set$
**syntax**
*-ctran* :: $'a\ conf \Rightarrow 'a\ conf \Rightarrow bool$ (- $-c\rightarrow$ - [81,81] 80)
*-ctran-∗*:: $'a\ conf \Rightarrow 'a\ conf \Rightarrow bool$ (- $-c{*}\rightarrow$ - [81,81] 80)
**translations**
$P\ -c\rightarrow\ Q\ \rightleftharpoons (P,Q) \in ctran$
$P\ -c{*}\rightarrow\ Q \rightleftharpoons (P,Q) \in ctran\hat{}{*}$

**inductive** *ctran*
**intros**
*Basic*: $(Some(Basic\ f),\ s)\ -c\rightarrow\ (None,\ f\ s)$

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

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

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

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

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

**monos** *rtrancl-mono*

### 3.2.2 Semantics of Parallel Programs

**types** $'a\ par\text{-}conf = ('a\ par\text{-}com) \times 'a$
**consts**
*par-etran* :: $('a\ par\text{-}conf \times 'a\ par\text{-}conf)\ set$
*par-ctran* :: $('a\ par\text{-}conf \times 'a\ par\text{-}conf)\ set$
**syntax**
*-par-etran*:: $['a\ par\text{-}conf,'a\ par\text{-}conf] \Rightarrow bool$ (- $-pe\rightarrow$ - [81,81] 80)
*-par-ctran*:: $['a\ par\text{-}conf,'a\ par\text{-}conf] \Rightarrow bool$ (- $-pc\rightarrow$ - [81,81] 80)
**translations**
$P\ -pe\rightarrow\ Q\ \rightleftharpoons (P,Q) \in par\text{-}etran$
$P\ -pc\rightarrow\ Q\ \rightleftharpoons (P,Q) \in par\text{-}ctran$

**inductive** *par-etran*

**intros**
*ParEnv*: $(Ps,\ s)\ -pe\rightarrow\ (Ps,\ t)$

**inductive** *par-ctran*
**intros**
*ParComp*: $[\![i<length\ Ps;\ (Ps!i,\ s)\ -c\rightarrow\ (r,\ t)]\!] \Longrightarrow (Ps,\ s)\ -pc\rightarrow\ (Ps[i:=r],\ t)$

### 3.2.3 Computations

#### Sequential computations

**types** *'a confs = ('a conf) list*
**consts** *cptn* :: *('a confs) set*
**inductive** *cptn*
**intros**
*CptnOne*: $[(P,s)] \in cptn$
*CptnEnv*: $(P,\ t)\#xs \in cptn \Longrightarrow (P,s)\#(P,t)\#xs \in cptn$
*CptnComp*: $[\![(P,s)\ -c\rightarrow\ (Q,t);\ (Q,\ t)\#xs \in cptn\ ]\!] \Longrightarrow (P,s)\#(Q,t)\#xs \in cptn$

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

#### Parallel computations

**types** *'a par-confs = ('a par-conf) list*
**consts** *par-cptn* :: *('a par-confs) set*
**inductive** *par-cptn*
**intros**
*ParCptnOne*: $[(P,s)] \in par\text{-}cptn$
*ParCptnEnv*: $(P,t)\#xs \in par\text{-}cptn \Longrightarrow (P,s)\#(P,t)\#xs \in par\text{-}cptn$
*ParCptnComp*: $[\![\ (P,s)\ -pc\rightarrow\ (Q,t);\ (Q,t)\#xs \in par\text{-}cptn\ ]\!] \Longrightarrow (P,s)\#(Q,t)\#xs \in par\text{-}cptn$

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

### 3.2.4 Modular Definition of Computation

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

**consts** *cptn-mod* :: *('a confs) set*
**inductive** *cptn-mod*
**intros**
*CptnModOne*: $[(P,\ s)] \in cptn\text{-}mod$
*CptnModEnv*: $(P,\ t)\#xs \in cptn\text{-}mod \Longrightarrow (P,\ s)\#(P,\ t)\#xs \in cptn\text{-}mod$

*CptnModNone*: $\llbracket$(*Some P*, *s*) $-c\rightarrow$ (*None*, *t*); (*None*, *t*)#*xs* $\in$ *cptn-mod* $\rrbracket \Longrightarrow$ (*Some P*,*s*)#(*None*, *t*)#*xs* $\in$*cptn-mod*
*CptnModCondT*: $\llbracket$(*Some P0*, *s*)#*ys* $\in$ *cptn-mod*; *s* $\in$ *b* $\rrbracket \Longrightarrow$ (*Some*(*Cond b P0 P1*), *s*)#(*Some P0*, *s*)#*ys* $\in$ *cptn-mod*
*CptnModCondF*: $\llbracket$(*Some P1*, *s*)#*ys* $\in$ *cptn-mod*; *s* $\notin$ *b* $\rrbracket \Longrightarrow$ (*Some*(*Cond b P0 P1*), *s*)#(*Some P1*, *s*)#*ys* $\in$ *cptn-mod*
*CptnModSeq1*: $\llbracket$(*Some P0*, *s*)#*xs* $\in$ *cptn-mod*; *zs*=*map* (*lift P1*) *xs* $\rrbracket$
$\Longrightarrow$ (*Some*(*Seq P0 P1*), *s*)#*zs* $\in$ *cptn-mod*
*CptnModSeq2*:
$\llbracket$(*Some P0*, *s*)#*xs* $\in$ *cptn-mod*; *fst*(*last* ((*Some P0*, *s*)#*xs*)) = *None*;
(*Some P1*, *snd*(*last* ((*Some P0*, *s*)#*xs*)))#*ys* $\in$ *cptn-mod*;
*zs*=(*map* (*lift P1*) *xs*)@*ys* $\rrbracket \Longrightarrow$ (*Some*(*Seq P0 P1*), *s*)#*zs* $\in$ *cptn-mod*

*CptnModWhile1*:
$\llbracket$ (*Some P*, *s*)#*xs* $\in$ *cptn-mod*; *s* $\in$ *b*; *zs*=*map* (*lift* (*While b P*)) *xs* $\rrbracket$
$\Longrightarrow$ (*Some*(*While b P*), *s*)#(*Some*(*Seq P* (*While b P*)), *s*)#*zs* $\in$ *cptn-mod*
*CptnModWhile2*:
$\llbracket$ (*Some P*, *s*)#*xs* $\in$ *cptn-mod*; *fst*(*last* ((*Some P*, *s*)#*xs*))=*None*; *s* $\in$ *b*;
*zs*=(*map* (*lift* (*While b P*)) *xs*)@*ys*;
(*Some*(*While b P*), *snd*(*last* ((*Some P*, *s*)#*xs*)))#*ys* $\in$ *cptn-mod*$\rrbracket$
$\Longrightarrow$ (*Some*(*While b P*), *s*)#(*Some*(*Seq P* (*While b P*)), *s*)#*zs* $\in$ *cptn-mod*

### 3.2.5 Equivalence of Both Definitions.

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

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

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

**lemma** *cptn-onlyif-cptn-mod* [*rule-format*]: *c* $\in$ *cptn* $\Longrightarrow$ *c* $\in$ *cptn-mod*
⟨*proof*⟩

**lemma** *lift-is-cptn*: *c*$\in$*cptn* $\Longrightarrow$ *map* (*lift P*) *c* $\in$ *cptn*
⟨*proof*⟩

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

**lemma** *last-lift*: $[\![xs \neq []; \mathit{fst}(xs!(\mathit{length}\ xs - (\mathit{Suc}\ 0))) = \mathit{None}]\!]$
$\Longrightarrow \mathit{fst}((\mathit{map}\ (\mathit{lift}\ P)\ xs)!(\mathit{length}\ (\mathit{map}\ (\mathit{lift}\ P)\ xs) - (\mathit{Suc}\ 0))) = (\mathit{Some}\ P)$
⟨*proof*⟩

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

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

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

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

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

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

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

**lemma** *cptn-if-cptn-mod*: $c \in \mathit{cptn\text{-}mod} \Longrightarrow c \in \mathit{cptn}$
⟨*proof*⟩

**theorem** *cptn-iff-cptn-mod*: $(c \in \mathit{cptn}) = (c \in \mathit{cptn\text{-}mod})$
⟨*proof*⟩

## 3.3 Validity of Correctness Formulas

### 3.3.1 Validity for Component Programs.

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

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

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

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

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

### 3.3.2 Validity for Parallel Programs.

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

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

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

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

### 3.3.3 Compositionality of the Semantics

#### Definition of the conjoin operator

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

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

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

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

$(c!j\ -pe\rightarrow\ c!Suc\ j \wedge (\forall i<length\ clist.\ (clist!i)!j\ -e\rightarrow\ (clist!i)!\ Suc\ j)))$

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

### Some previous lemmas

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**lemma** *tl-zero*[*rule-format*]:
$P$ $(ys!Suc\ j)$ $\longrightarrow$ $Suc\ j<length\ ys$ $\longrightarrow$ $ys\neq[]$ $\longrightarrow$ $P$ $(tl(ys)!j)$
⟨*proof*⟩

### 3.3.4 The Semantics is Compositional

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

**lemma** *less-Suc-0* [*iff*]: $(n < Suc\ 0) = (n = 0)$
⟨*proof*⟩

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

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

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

**end**

## 3.4 The Proof System

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

### 3.4.1 Proof System for Component Programs

**declare** *Un-subset-iff* [*iff del*]

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

**consts** $rghoare :: ('a\ rgformula)\ set$
**syntax**
$\text{-}rghoare :: ['a\ com,\ 'a\ set,\ ('a \times 'a)\ set,\ ('a \times 'a)\ set,\ 'a\ set] \Rightarrow bool$
$(\vdash\ \text{-}\ sat\ [\text{-},\ \text{-},\ \text{-},\ \text{-}]\ [60,0,0,0,0]\ 45)$
**translations**
$\vdash P\ sat\ [pre,\ rely,\ guar,\ post] \rightleftharpoons (P,\ pre,\ rely,\ guar,\ post) \in rghoare$

**inductive** *rghoare*
**intros**
*Basic*: $[\![\ pre \subseteq \{s.\ f\ s \in post\};\ \{(s,t).\ s \in pre \wedge (t{=}f\ s \vee t{=}s)\} \subseteq guar;$
$stable\ pre\ rely;\ stable\ post\ rely\ ]\!]$
$\Longrightarrow \vdash Basic\ f\ sat\ [pre,\ rely,\ guar,\ post]$

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

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

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

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

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

**constdefs**
$Pre :: 'a\ rgformula \Rightarrow 'a\ set$
$Pre\ x \equiv fst(snd\ x)$
$Post :: 'a\ rgformula \Rightarrow 'a\ set$
$Post\ x \equiv snd(snd(snd(snd\ x)))$
$Rely :: 'a\ rgformula \Rightarrow ('a \times 'a)\ set$
$Rely\ x \equiv fst(snd(snd\ x))$
$Guar :: 'a\ rgformula \Rightarrow ('a \times 'a)\ set$
$Guar\ x \equiv fst(snd(snd(snd\ x)))$

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

### 3.4.2 Proof System for Parallel Programs

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

**consts** $par\text{-}rghoare :: ('a\ par\text{-}rgformula)\ set$
**syntax**
$\text{-}par\text{-}rghoare :: ('a\ rgformula)\ list \Rightarrow 'a\ set \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set \Rightarrow 'a\ set \Rightarrow bool \quad (\vdash\ \text{-}\ SAT\ [\text{-}, \text{-}, \text{-}, \text{-}]\ [60,0,0,0,0]\ 45)$
**translations**
$\vdash Ps\ SAT\ [pre, rely, guar, post] \rightleftharpoons (Ps, pre, rely, guar, post) \in par\text{-}rghoare$

**inductive** $par\text{-}rghoare$
**intros**
$Parallel$:
$[\![\ \forall i{<}length\ xs.\ rely \cup (\bigcup j{\in}\{j.\ j{<}length\ xs \wedge j{\neq}i\}.\ Guar(xs!j)) \subseteq Rely(xs!i);$
$(\bigcup j{\in}\{j.\ j{<}length\ xs\}.\ Guar(xs!j)) \subseteq guar;$
$pre \subseteq (\bigcap i{\in}\{i.\ i{<}length\ xs\}.\ Pre(xs!i));$
$(\bigcap i{\in}\{i.\ i{<}length\ xs\}.\ Post(xs!i)) \subseteq post;$
$\forall i{<}length\ xs.\ \vdash Com(xs!i)\ sat\ [Pre(xs!i),Rely(xs!i),Guar(xs!i),Post(xs!i)]\ ]\!]$
$\Longrightarrow\ \vdash xs\ SAT\ [pre, rely, guar, post]$

## 3.5 Soundness

### Some previous lemmas

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

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

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

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

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

$\langle proof \rangle$

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

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

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

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

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

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

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

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

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

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

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

**lemma** *stability* [*rule-format*]:
$\forall j\ k.\ x \in cptn \longrightarrow stable\ p\ rely \longrightarrow j{\leq}k \longrightarrow k{<}length\ x \longrightarrow snd(x!j){\in}p\ \longrightarrow$

$(\forall i.\ (Suc\ i){<}length\ x \longrightarrow$
$\quad (x!i\ {-e\rightarrow}\ x!(Suc\ i)) \longrightarrow (snd(x!i),\ snd(x!(Suc\ i))) \in rely) \longrightarrow$
$(\forall i.\ j{\le}i \wedge i{<}k \longrightarrow x!i\ {-e\rightarrow}\ x!Suc\ i) \longrightarrow snd(x!k){\in}p \wedge fst(x!j){=}fst(x!k)$
$\langle proof\rangle$

### 3.5.1 Soundness of the System for Component Programs

#### Soundness of the Basic rule

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

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

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

#### Soundness of the Await rule

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

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

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

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

$\Longrightarrow \models$ *Await b P sat* [*pre*, *rely*, *guar*, *post*]
⟨*proof*⟩

### Soundness of the Conditional rule

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

### Soundness of the Sequential rule

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

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

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

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

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

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

### Soundness of the While rule

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

**lemma** *assum-after-body*:

$\llbracket \models P\ sat\ [pre \cap b,\ rely,\ guar,\ pre];$
$(Some\ P,\ s)\ \#\ xs \in cptn\text{-}mod;\ fst\ (last\ ((Some\ P,\ s)\ \#\ xs)) = None;\ s \in b;$
$(Some\ (While\ b\ P),\ s)\ \#\ (Some\ (Seq\ P\ (While\ b\ P)),\ s)\ \#$
$map\ (lift\ (While\ b\ P))\ xs\ @\ ys \in assum\ (pre,\ rely)\rrbracket$
$\Longrightarrow (Some\ (While\ b\ P),\ snd\ (last\ ((Some\ P,\ s)\ \#\ xs)))\ \#\ ys \in assum\ (pre,\ rely)$
$\langle proof \rangle$

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

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

### Soundness of the Rule of Consequence

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

### Soundness of the system for sequential component programs

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

## 3.5.2 Soundness of the System for Parallel Programs

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

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

⟨*proof*⟩

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

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

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

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

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

**theorem** *par-rgsound*:
$\vdash c\ SAT\ [pre,\ rely,\ guar,\ post] \Longrightarrow$

$\models$ (*ParallelCom c*) *SAT* [*pre*, *rely*, *guar*, *post*]
⟨*proof*⟩

**end**

## 3.6 Concrete Syntax

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

**syntax**
*-Assign* :: *idt* ⇒ ′*b* ⇒ ′*a com* ((´- :=/ -) [*70*, *65*] *61*)
*-skip* :: ′*a com* (*SKIP*)
*-Seq* :: ′*a com* ⇒ ′*a com* ⇒ ′*a com* ((-;;/ -) [*60*,*61*] *60*)
*-Cond* :: ′*a bexp* ⇒ ′*a com* ⇒ ′*a com* ⇒ ′*a com* ((*0IF* -/ *THEN* -/ *ELSE* -/*FI*) [*0*, *0*, *0*] *61*)
*-Cond2* :: ′*a bexp* ⇒ ′*a com* ⇒ ′*a com* ((*0IF* - *THEN* - *FI*) [*0*,*0*] *56*)
*-While* :: ′*a bexp* ⇒ ′*a com* ⇒ ′*a com* ((*0WHILE* - /*DO* - /*OD*) [*0*, *0*] *61*)
*-Await* :: ′*a bexp* ⇒ ′*a com* ⇒ ′*a com* ((*0AWAIT* - /*THEN* /- /*END*) [*0*,*0*] *61*)
*-Atom* :: ′*a com* ⇒ ′*a com* ((⟨-⟩) *61*)
*-Wait* :: ′*a bexp* ⇒ ′*a com* ((*0WAIT* - *END*) *61*)

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

**nonterminals**
*prgs*

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

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

**syntax**
  *-prg-scheme* :: $['a,\ 'a,\ 'a,\ 'a] \Rightarrow prgs$  $(SCHEME\ [\text{-} \leq \text{-} < \text{-}]\ \text{-}\ [0,0,0,60]\ 57)$

**translations**
  *-prg-scheme* $j\ i\ k\ c \rightleftharpoons (map\ (\lambda i.\ c)\ [j..<k])$

Translations for variables before and after a transition:

**syntax**
  *-before* :: $id \Rightarrow 'a$ ($^{\text{o}}$-)
  *-after* :: $id \Rightarrow 'a$ ($^{\text{a}}$-)

**translations**
  $^{\text{o}}x \rightleftharpoons x\ \acute{}fst$
  $^{\text{a}}x \rightleftharpoons x\ \acute{}snd$

⟨*ML*⟩

**end**

## 3.7 Examples

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

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

### 3.7.1 Set Elements of an Array to Zero

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

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

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

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

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

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

### 3.7.2 Increment a Variable in Parallel

#### Two components

**record** *Example2* =
$x$ :: *nat*
*c-0* :: *nat*
*c-1* :: *nat*

**lemma** *Example2*:
$\vdash$ *COBEGIN*
$(\langle\ ´x := ´x+1;;\ ´c\text{-}0 := ´c\text{-}0 + 1\ \rangle,$
$\{\!|´x = ´c\text{-}0 + ´c\text{-}1 \ \wedge\ ´c\text{-}0 = 0|\!\},$
$\{\!|{}^{o}c\text{-}0 = {}^{a}c\text{-}0 \wedge$
$({}^{o}x = {}^{o}c\text{-}0 + {}^{o}c\text{-}1$
$\longrightarrow {}^{a}x = {}^{a}c\text{-}0 + {}^{a}c\text{-}1)|\!\},$
$\{\!|{}^{o}c\text{-}1 = {}^{a}c\text{-}1 \wedge$
$({}^{o}x = {}^{o}c\text{-}0 + {}^{o}c\text{-}1$
$\longrightarrow {}^{a}x = {}^{a}c\text{-}0 + {}^{a}c\text{-}1)|\!\},$
$\{\!|´x = ´c\text{-}0 + ´c\text{-}1 \wedge ´c\text{-}0 = 1\ |\!\})$
$\|$
$(\langle\ ´x := ´x+1;;\ ´c\text{-}1 := ´c\text{-}1+1\ \rangle,$
$\{\!|´x = ´c\text{-}0 + ´c\text{-}1 \wedge ´c\text{-}1 = 0\ |\!\},$
$\{\!|{}^{o}c\text{-}1 = {}^{a}c\text{-}1 \wedge$
$({}^{o}x = {}^{o}c\text{-}0 + {}^{o}c\text{-}1$
$\longrightarrow {}^{a}x = {}^{a}c\text{-}0 + {}^{a}c\text{-}1)|\!\},$
$\{\!|{}^{o}c\text{-}0 = {}^{a}c\text{-}0 \wedge$
$({}^{o}x = {}^{o}c\text{-}0 + {}^{o}c\text{-}1$
$\longrightarrow {}^{a}x = {}^{a}c\text{-}0 + {}^{a}c\text{-}1)|\!\},$
$\{\!|´x = ´c\text{-}0 + ´c\text{-}1 \wedge ´c\text{-}1 = 1|\!\})$
*COEND*

$SAT\ [\{\!|\ ´x{=}0 \wedge ´c\text{-}0{=}0 \wedge ´c\text{-}1{=}0 |\!\},$
$\quad \{\!|\ {}^{o}x{=}{}^{a}x \wedge {}^{o}c\text{-}0{=}\ {}^{a}c\text{-}0 \wedge {}^{o}c\text{-}1{=}{}^{a}c\text{-}1 |\!\},$
$\quad \{\!|\ True |\!\},$
$\quad \{\!|\ ´x{=}2 |\!\}]$
$\langle proof\rangle$

### Parameterized

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

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

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

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

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

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

### 3.7.3 Find Least Element

A previous lemma:

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

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

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

Same but with a list as auxiliary variable:

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

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

**end**