# Mechanical Generation of Invariants for FOR-Loops


Stefan Kauer
BU DE
EADS Deutschland GmbH
Immenstaad, Germany

Jürgen F H Winkler
Institute of Informatics
Friedrich Schiller University
Jena, Germany



Course on “Mechanical Program Verification”
Budapest, ELTE, 08-12 Oct 2007

# Overview

- Basic Approach
- Bound Transformation
- Determination of Common Conjuncts
- Adaptation of Proof Rules

# Basic Approach

- A specification $S$ is given
- A program $P$ has been developed
- Correctness $\equiv$ P conforms to S
- Proof of correctness should be done mechanically
  (free the SW engineer from tedious work)
- VCs based on $wp(\cdot,\cdot)$

# Computation of wp(·,·)

- For assignment, IF, CASE, Sequence: simple formula manipulation
- For WHILE
  - find a solution for H (e.g. Kauer) ( $\langle \exists k: 0 \leq k: H_k(post) \rangle$ )
  - use a VC based on (pre, inv, post, term)
- For FOR: use a VC based on (pre, inv, post)

# FOR-loop

```
      -- pre
FOR  i  IN  LO .. UP
LOOP  Body
      -- inv
END LOOP
      -- post
```

i:  loop variable: is a constant in BODY

strictly controlled loop

LO:  lower bound: no side effects and referentially transparent in BODY

UP:  upper bound : no side effects and referentially transparent in BODY

holds for many FOR-loops in practice  (see summary)

# FOR-loop: VC

$[\ PRE \Rightarrow LO, UP \in Ti\ ] \ \wedge$

$[\ PRE \wedge LO > UP \Rightarrow POST\ ] \ \wedge$

$[\ PRE \wedge LO \leq UP \Rightarrow wp(\ BODY^{i}_{LO}, INV^{i}_{LO}\ )\ ] \ \wedge$

$[\ LO \leq i < UP \wedge INV \Rightarrow wp(\ BODY^{i}_{i+1}, INV^{i}_{i+1}\ )\ ] \ \wedge$

$[\ LO \leq UP \wedge INV^{i}_{UP} \Rightarrow POST\ ]$

Ti: set of admissible values of i

# FOR-loop: example

-- PRE: $s=0 \wedge 0 \le n \le 65535 \wedge n=N$

```
FOR i  in  1..n  LOOP      s := s+i;
END LOOP
```

-- POST: $s = \langle \Sigma j\text{: } 1..n\text{: } j \rangle \wedge 0 \le n \le 65535 \wedge n=N$

# Verification of a FOR-loop requires PRE, POST and INV

=> spec gives PRE and POST

=> program developer must find/invent INV

=> easier if INV can be computed by the program prover

# How to compute INV?

Observation by Dijkstra and Gries for WHILE-loops:

[Dij 1976; Grie 1982]

=> INV can be seen as a weakening of POST

=> one weakening: replace a constant in POST by a variable (RCPV)

Example [Gri 1983: 199]

POST $\equiv$ sum = $\langle \Sigma j$: $0 \le j < n$: $b(j) \rangle$ -- n is constant for this possibly iterative
-- process

HI $\equiv$ sum = $\langle \Sigma j$: $0 \le j < i$: $b(j) \rangle$ -- i is a variable, possibly the loop
-- variable

Used in the development of the program from (PRE, POST)

# How to compute INV?

Kauer for FOR-loop and verification of an existing loop against (PRE, POST):

-- PRE: $s = 0 \ \wedge \ s \in$ int32
FOR i in 1..100 LOOP s := s+i;
  -- HI: $s = \langle \Sigma j\!: 1..i\!: j \rangle \ \wedge \ s \in$ int32
END LOOP
  -- POST: $s = \langle \Sigma j\!: 1..100\!: j \rangle \ \wedge \ s \in$ int32

Use UP as candidate for the constant to be replaced in POST

$$\text{HI} \equiv \text{POST}^{100}_{i} \equiv (s = \langle \Sigma j\!: 1..100\!: j \rangle \ \wedge \ s \in \text{int32})^{100}_{i}$$
$$\equiv s = \langle \Sigma j\!: 1..i\!: j \rangle \ \wedge \ s \in \text{int32}$$ is an invariant

# How to compute INV?

Naïve RCPV does often not work:

-- PRE: $s=0 \wedge 0 \le n \le 65535 \wedge n=N$

FOR i in 1..n LOOP s := s+i; END LOOP

-- POST: $s=\langle\Sigma j: 1..n: j\rangle \wedge 0 \le n \le 65535 \wedge n=N$

$HI \equiv POST^{n}_{i} \equiv (s=\langle\Sigma j: 1..n: j\rangle \wedge 0 \le n \le 65535 \wedge n=N)^{n}_{i}$

$\equiv s=\langle\Sigma j: 1..i: j\rangle \wedge 0 \le i \le 65535 \wedge i=N$

Since N is a constant HI cannot be an invariant

=> apply RCPV not to common conjuncts

$HI' \equiv (s=\langle\Sigma j: 1..n: j\rangle)^{n}_{i} \wedge 0 \le n \le 65535 \wedge n=N$

$\equiv s=\langle\Sigma j: 1..i: j\rangle \wedge 0 \le n \le 65535 \wedge n=N$ is an invariant

## Method up to now only applicable if final bound is a simple variable

```
    -- PRE: s = 0
FOR  i  in 1..n+m  LOOP  s := s+i;  END LOOP
    -- POST: s = ⟨Σj: 1..n+m: j⟩
```

First idea: auxiliary variable for UP

```
    -- PRE: s = 0
vup := n+m;                          --  fresh variable
    --  s = 0 ∧ vup = n+m            --  sp(s=0, "vup := n+m;")
FOR i in 1..vup LOOP s := s+i; END LOOP
    -- POST: s = ⟨Σj: 1..n+m: j⟩
```

HI $\equiv$ POST because vup does not occur in POST

HI is not an invariant

## Method up to now only applicable if final bound is a simple variable

Second idea: bound transformation: t(UP,n)(e) = e – m (translation)

t(UP,n)(UP) = n+m-m = n is a simple variable

Compensation in BODY: $\text{BODY}^{i}_{i+m}$

```
        -- PRE: s = 0
 -- FOR  i  in 1..n+m  LOOP  s := s+i;   END LOOP
    FOR  i  in 1-m..n   LOOP  s := s+(i+m);  END LOOP
    -- POST: s = ⟨Σj: 1..n+m: j⟩  ≡  s = ⟨Σj: 1-m..n: j+m⟩
```

## Original and transformed loop

```
L1:    -- PRE
       FOR i in  LO1..UP1  LOOP  BODY  END LOOP
         -- POST

L2:    -- PRE
       FOR i in  t(LO1) .. r(t(UP1))
       LOOP  BODY(i ↦ t*(i))  END LOOP
         -- POST
```

# Translation functions

We deal with translations $t(UP)$ such that

$r(t(UP))$ is "$v$" or "$-v$" for some $v \in free(UP)$

"v": $HI = POST^{v}_{i}$

"-v": $HI = POST^{v}_{(-i)}$

For a given UP there may exist several translations:

$UP = n+m$: $t1(UP,m)(e) = e-n$, $t2(UP,n)(e) = e-m$

## Translation functions

HI is really hypothetical

```
-- PRE: s = 0 ∧ m ≥ 0 ∧ n≤0
  FOR i IN m .. m-n LOOP
    s := s + i;
  END LOOP;
    -- POST: s = ⟨Σ j: m..m-n: j⟩
```

t1(m-n,m)(e) = e+n: HI $\equiv$ $POST^{m}_{i}$ $\equiv$ $s = \langle \Sigma\ j: i..i-n: j\rangle$ is not an invariant

t2(m-n,n)(e) = e-m: HI $\equiv$ $POST^{n}_{(-i)}$ $\equiv$ $s = \langle \Sigma\ j: m..m+i: j\rangle$ is an invariant

# Translation functions

*Form of UP*

| | o1 v | e1 o2 v | o1 v o2 e1 | e1 o2 v o3 e2 |
|---|---|---|---|---|
| t(UP,v)(e) | e | e-e1 | e $o2^{-1}$ e1 | e - e1 $o3^{-1}$ e2 |
| r(t(UP,v)(UP)) | o1 v | o2 v | o1 v | o2 v |
| t*(UP,v)(e) | e | e+e1 | e o2 e1 | e + e1 o3 e2 |
| t*(UP,v)(t(UP,v)(e)) | e | e-e1+e1 | e $o2^{-1}$ e1 o2 e1 | e - e1 $o3^{-1}$ e2 + e1 o3 e2 |

$v \notin \text{free}(e1) \cup \text{free}(e2)$, $o1 \in \{+, -, \varepsilon\}$, $o2, o3 \in \{+, -\}$, $+^{-1} = -$ , $-^{-1} = +$

For these translations the transformed loop is executed for the same sequence of values of the loop variable as the original loop

# Determination of Common Conjuncts

(a) transform PRE and POST into normal form NF

(b) determine the syntactically common conjuncts $C = C_1 \wedge \ldots \wedge C_n$

(c) determine those $C_i$ for which

$$\text{noWrite(BODY, free}(C_i)) \vee [C_i \Rightarrow \text{wp(BODY}, C_i)] \quad \text{holds}$$

$C_{\mathbf{com}}$ is the conjunctions of these $C_i$

$$\text{POST} \equiv \text{POST}' \wedge C_{\mathbf{com}}$$

# Specialized Proof Rules

(upwards counting, bounds not modified, “v”, i $\notin$ free(POST)) :

$[\ PRE \Rightarrow LO, UP \in Ti\ ] \ \wedge$

$[\ PRE \wedge LO > UP \Rightarrow POST'\ ] \ \wedge$

$[\ PRE \wedge LO \le UP \Rightarrow wp(\ BODY^{i}_{LO}, POST'^{r(t(UP))}_{t(LO)}\ )\ ] \ \wedge$

$[\ t(LO) \le i < t(UP) \wedge POST'^{r(t(UP))}_{i} \wedge C_{com} \Rightarrow$

$\quad wp(BODY^{i}_{t^*(i)+1}, POST'^{r(t(UP))}_{i+1}\ )\ ]$

# Summary

- mechanical derivation of an hypothetical invariant from UP and POST

  based on RCPV, translation functions, identification of common conjuncts

- it is a heuristic and not a general solution

BUT: applicable to many practical FOR-loops:

BG 91: Gonnet, G. H.; Baeza-Yates, R.: Handbook of Algorithms and Data Structures. Addison Wesley, Wokingham, 1991:

in most FOR-loops (in this book) UP has one of the 3 forms:
(a) variable,
(b) sum of two variables
(c) sum of a variable and a constant

these are already covered by the method

# Summary

Randwertproblemlöser (boundary value problem solver)

Fortran-program written by Hermann and Kaiser of FSU Dept. Math&CS

1015 FOR-loops (DO-loop in Fortran)

998 of these loops are appropriate for the method

=> this heuristic method seems to be quite good

- more translation schemes could be defined
- not yet implemented

# *Thank You*

# Mechanical Generation of Invariants for FOR-Loops


Stefan Kauer
BU DE
EADS Deutschland GmbH
Immenstaad, Germany

Jürgen F H Winkler
Institute of Informatics
Friedrich Schiller University
Jena, Germany



WING 2007
RISC, Hagenberg, Austria
2007.Jun.26

# Specification and Program

- Specification is given by assertions: S = (pre, post)

pre

P

post

Verification condition $VC \equiv [pre \Rightarrow wp(P, post)]$ or

$VC \equiv [sp(pre, P) \Rightarrow post]$

# Compositionality of wp(·,·)

- $wp(S1\ S2, post) \equiv wp(S1, wp(S2, post))$
- $wp(\text{if } C \text{ then } S1 \text{ else } S2 \text{ fi}, post) \equiv$
  $\text{well-defined}(C) \text{ cand } ( (C \Rightarrow wp(S1, post) ) \wedge$
  $(\neg C \Rightarrow wp((S2, post) )$
- $wp(\text{while } C \text{ do } S \text{ od}, post) \equiv$
  $\text{well-defined}(C) \text{ cand } ( (C \Rightarrow wp(S\ \ WHILE, post) \wedge$
  $(\neg C \Rightarrow post) )$
- $wp(\text{while } C \text{ do } S \text{ od}, post) \equiv$ (VCW)
  $\langle \exists k: 0 \leq k: H_k(post) \rangle$