Automatic Checking of Invariant Diagrams

Johannes Eriksson
(joint work with Ralph-Johan Back and Magnus Myreen[1])

Åbo Akademi University, Turku, Finland

January 30, 2010

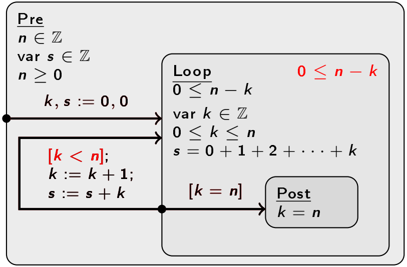
[1] University of Cambridge

Overview

- Invariant based programming revisited
- Socos tool: constructing a verified sorting program
- Ongoing work

Invariant Based Programming

Correct-by-construction method developed by Ralph Back.



Pre
$n \in \mathbb{Z}$
var $s \in \mathbb{Z}$
$n \geq 0$

$k, s := 0, 0$

Loop
$0 \leq n - k$
var $k \in \mathbb{Z}$
$0 \leq k \leq n$
$s = 0 + 1 + 2 + \cdots + k$

$0 \leq n - k$

$[k < n]$;
$k := k + 1$;
$s := s + k$

$[k = n]$

Post
$k = n$

$$
\begin{aligned}
& n \in \mathbb{Z} \wedge s \in \mathbb{Z} \wedge n \geq 0 \wedge k' = 0 \wedge s' = 0 \\
\Longrightarrow \quad & n \in \mathbb{Z} \wedge s' \in \mathbb{Z} \wedge n \geq 0 \wedge k' \in \mathbb{Z} \wedge \\
& 0 \leq k' \leq n \wedge s' = 0 + 1 + \cdots + k' \qquad ☑
\end{aligned}
$$

$$
\begin{aligned}
& n \in \mathbb{Z} \wedge s \in \mathbb{Z} \wedge n \geq 0 \wedge \\
& k \in \mathbb{Z} \wedge 0 \leq k \leq n \wedge s = 0 + 1 + \cdots + k \wedge \\
& k < n \wedge k' = k + 1 \wedge s' = s + k' \\
\Longrightarrow \quad & n \in \mathbb{Z} \wedge s' \in \mathbb{Z} \wedge n \geq 0 \wedge k' \in \mathbb{Z} \wedge \\
& 0 \leq k' \leq n \wedge s' = 0 + 1 + \cdots + k' \qquad ☑
\end{aligned}
$$

$$
\begin{aligned}
& n \in \mathbb{Z} \wedge s \in \mathbb{Z} \wedge n \geq 0 \wedge \\
& k \in \mathbb{Z} \wedge 0 \leq k \leq n \wedge s = 0 + 1 + \cdots + k \wedge \\
& k < n \wedge k' = k + 1 \wedge s' = s + k' \\
\Longrightarrow \quad & 0 \leq n - k' < n - k \wedge \\
& (k < n \vee k = n) \qquad ☑
\end{aligned}
$$

$$
\begin{aligned}
& n \in \mathbb{Z} \wedge s \in \mathbb{Z} \wedge n \geq 0 \wedge \\
& k \in \mathbb{Z} \wedge 0 \leq k \leq n \wedge s = 0 + 1 + \cdots + k \wedge \\
& k = n \\
\Longrightarrow \quad & n \in \mathbb{Z} \wedge s \in \mathbb{Z} \wedge n \geq 0 \wedge k \in \mathbb{Z} \wedge \\
& 0 \leq k \leq n \wedge s = 0 + 1 + \cdots + k \wedge \\
& k = n \qquad ☑
\end{aligned}
$$

Socos environment

A prototype tool to support invariant based programming.

- provides a diagram editor;
- automates generation of verification conditions;
- simplifies them with SMT solvers;
- gives immediate feedback;
- (also: compiles)

User works in the diagrammatic environment; no theorem prover round-trip.

- the diagram is the proof!
- usable without profiency in theorem provers (e.g., by students)

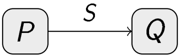
(demo)

Verification method

Transitions are reduced to VCs by applying the weakest precondition:

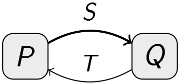
- **Consistency**:

  $P \Rightarrow \mathrm{wp}(S)(Q)$



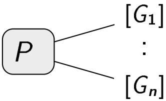
- **Termination**:

  $$(V = V_0) \wedge P \Rightarrow \mathrm{wp}(S)(V < V_0)$$
  $$(V = V_0) \wedge Q \Rightarrow \mathrm{wp}(T)(V \leq V_0)$$



- **Liveness**:

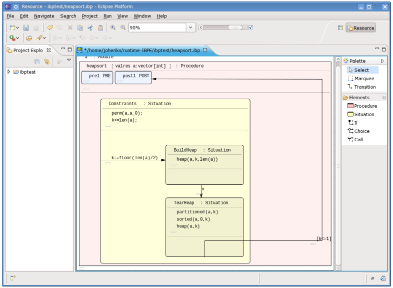
  $$P \Rightarrow G_1 \vee \cdots \vee G_n$$



Currently finishing new version of the Socos tool...

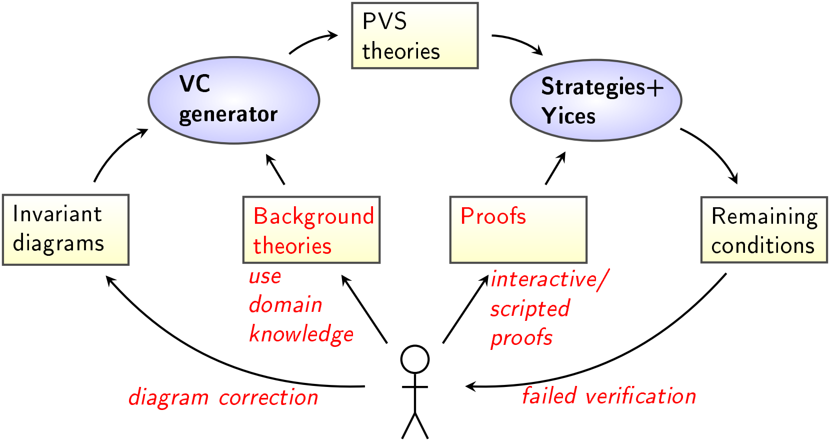
Features:

- Uses **PVS** as specification and implementation language
- **Fine grained** checking
- Uses **Yices** to discharge VCs
- **Eclipse** based front-end:


Resource - ibptest/heapsort.ibp - Eclipse Platform
File Edit Navigate Search Project Run View Window Help
90%
Resource
Project Explo
ibptest
*/home/joheriks/runtime-IBPE/ibptest/heapsort.ibp
Palette
Select
Marquee
Transition
Elements
Procedure
Situation
If
Choice
Call

```
heapsort [ valres a:vector[int] ] : Procedure
pre1 PRE
post1 POST
Constraints : Situation
perm(a,a_0);
k<=len(a);
k:=floor(len(a)/2)
BuildHeap : Situation
heap(a,k,len(a))
x
TearHeap : Situation
partitioned(a,k)
sorted(a,0,k)
heap(a,k)
[k<=1]
```

Architecture & workflow



PVS theories

VC generator

Strategies+Yices

Invariant diagrams

Background theories

Proofs

Remaining conditions

use domain knowledge

interactive/ scripted proofs

diagram correction

failed verification