ACL2 Code Proofs

J Strother Moore
Fall, 2013

Lecture 3

Today

partial correctness

partial correctness and clocks

Dave Greve's wormhole abstraction

inductive invariant style proofs

State Based Partial Correctness

$$((\texttt{pre}\ s)\ \wedge\ (\texttt{haltedp}\ (\texttt{m1}\ s\ k)))\ \rightarrow\ (\texttt{post}\ s\ (\texttt{m1}\ s\ k))$$

Demo

Clock-functions can be used to prove such theorems.

In this demo, we'll prove a non-termination result, which is a special case of partial correctness:

$$
\begin{array}{l}
((\texttt{pre}\ s) \\
\ \wedge \\
\ (\texttt{haltedp}\ (\texttt{m1}\ s\ k))) \\
\rightarrow \\
\ (\texttt{post}\ s\ (\texttt{m1}\ s\ k))
\end{array}
$$

Demo

Clock-functions can be used to prove such theorems.

In this demo, we'll prove a non-termination result, which is a special case of partial correctness:

$$((\texttt{pre}\ s)\ \wedge\ (\texttt{haltedp}\ (\texttt{m1}\ s\ k)))\ \rightarrow\ (\texttt{post}\ s\ (\texttt{m1}\ s\ k))$$

Demo

Clock-functions can be used to prove such theorems.

In this demo, we'll prove a non-termination result, which is a special case of partial correctness:

$$
\begin{array}{l}
(\texttt{(pre}\ s\texttt{)} \\
\ \wedge \\
\ \neg\texttt{(post}\ s\ \texttt{(m1}\ s\ k\texttt{)))} \\
\rightarrow \\
\ \neg\texttt{(haltedp (m1}\ s\ k\texttt{))}
\end{array}
$$

Demo

Clock-functions can be used to prove such theorems.

In this demo, we'll prove a non-termination result, which is a special case of partial correctness:

$$\begin{array}{l} (\texttt{(pre}\ s\texttt{)} \\ \ \wedge \\ \ \neg\texttt{(post}\ s\ \texttt{(m1}\ s\ k\texttt{)))} \\ \rightarrow \\ \ \neg\texttt{(haltedp (m1}\ s\ k\texttt{))} \end{array}$$

Demo

Clock-functions can be used to prove such theorems.

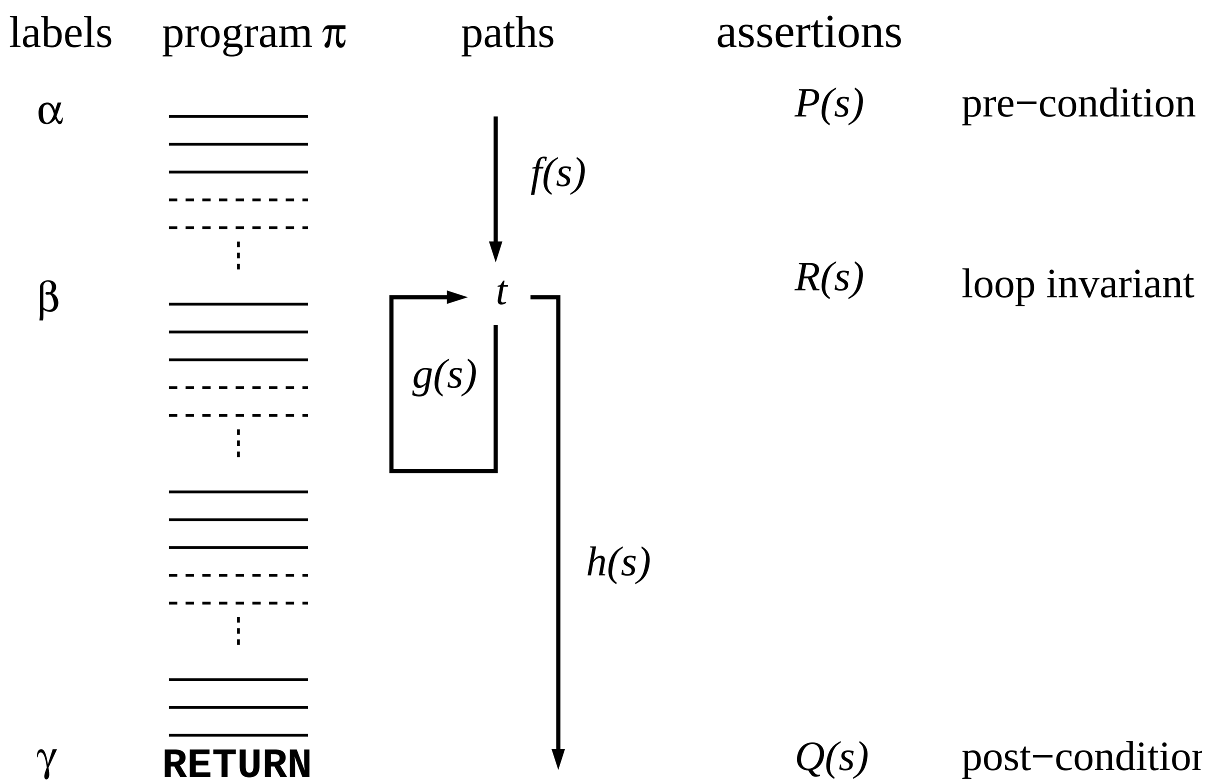
In this demo, we'll prove a non-termination result, which is a special case of partial correctness:

$$(\texttt{pre}\ s)$$

$$\rightarrow$$

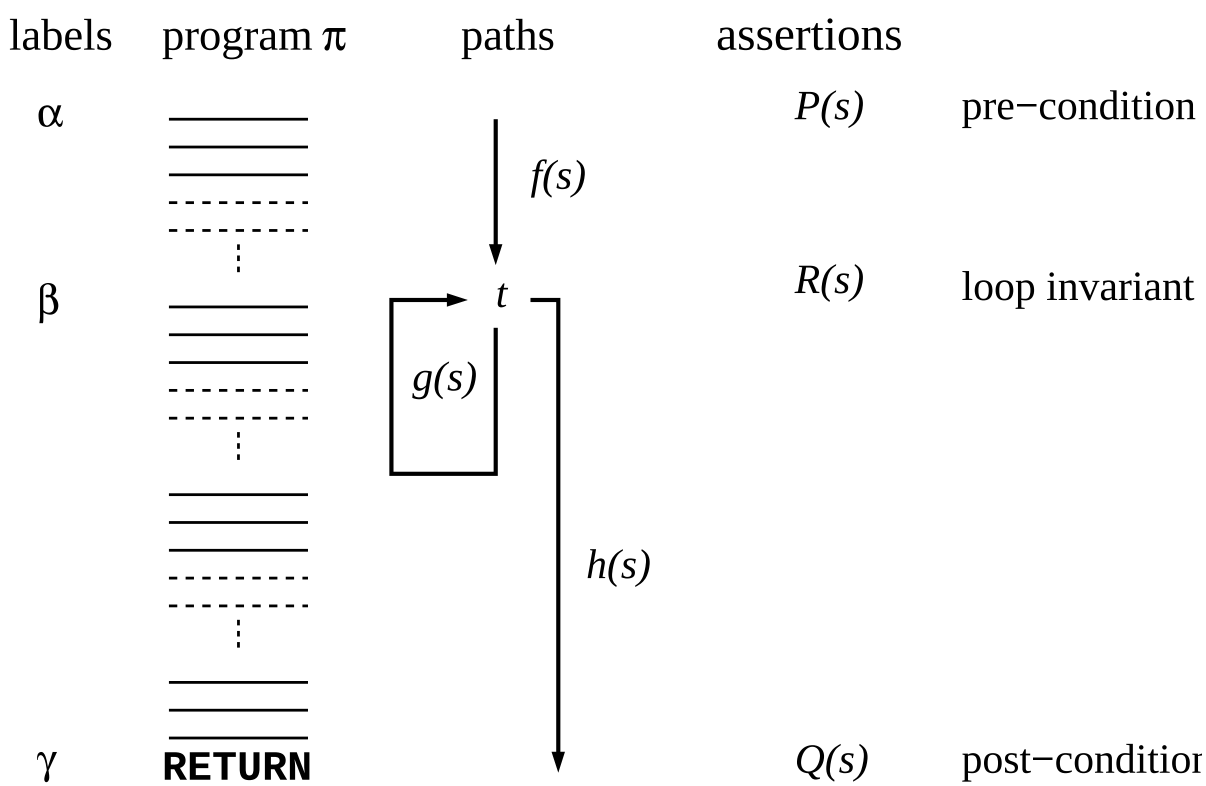
$$\neg\texttt{(haltedp (m1}\ s\ k\texttt{))}$$

Conventional Mechanized Code Proofs



VC1. $P(s) \rightarrow R(f(s))$,

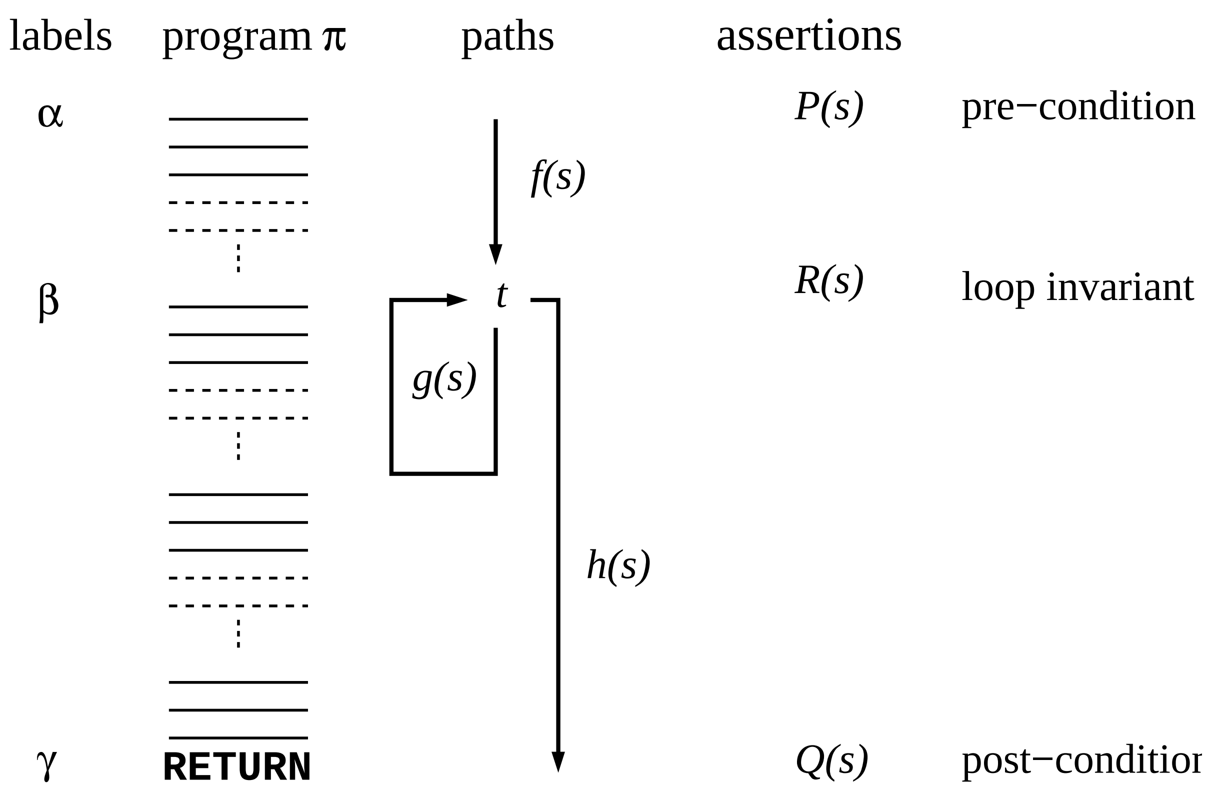
Conventional Mechanized Code Proofs



VC1. $P(s) \rightarrow R(f(s))$,

VC2. $R(s) \wedge t \rightarrow R(g(s))$, and

Conventional Mechanized Code Proofs



VC1. $P(s) \to R(f(s))$,

VC2. $R(s) \wedge t \to R(g(s))$, and

VC3. $R(s) \wedge \neg t \to Q(h(s))$.

Conventional Mechanized Code Proofs

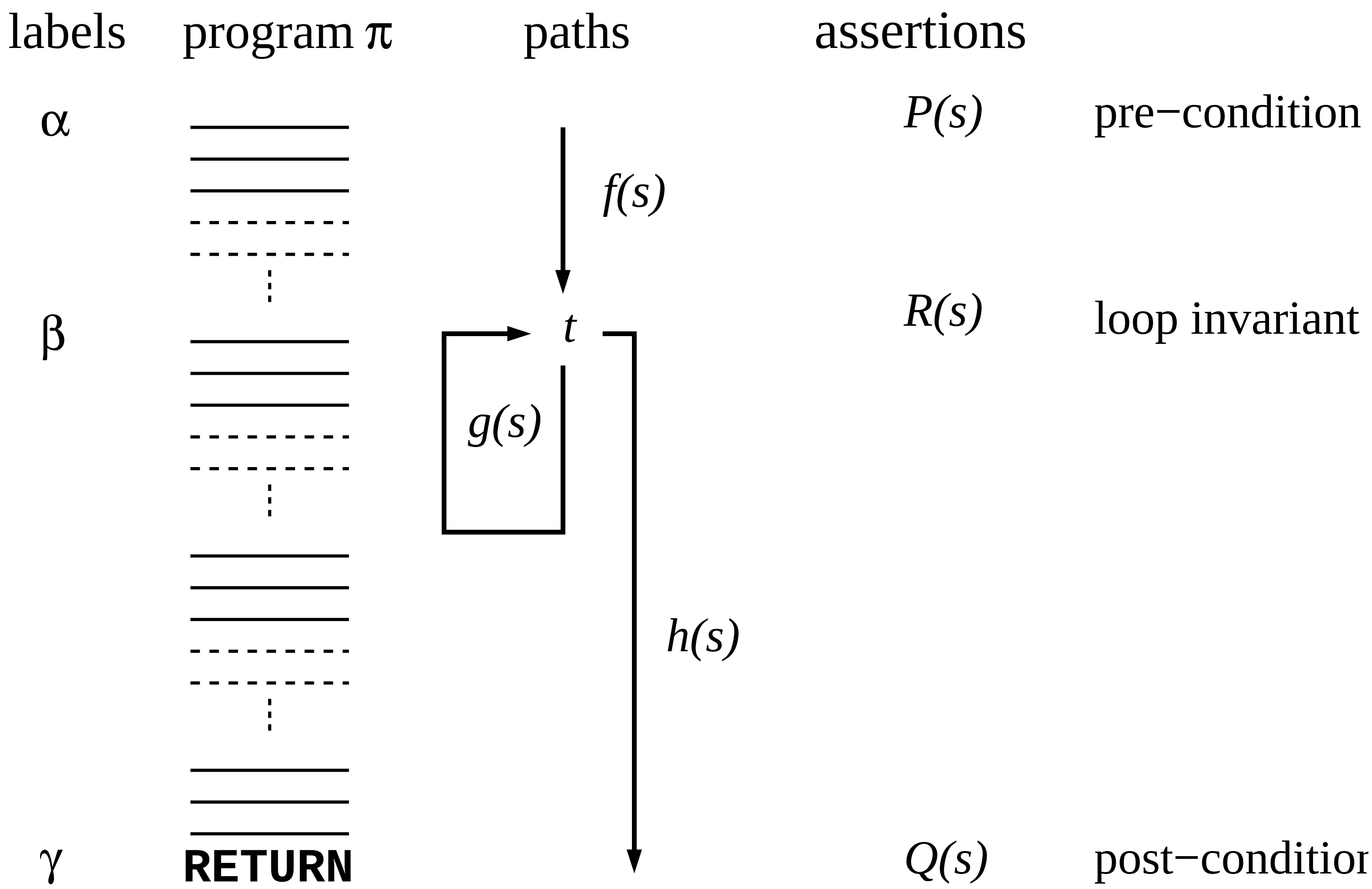
Code is annotated with assertions.

A special-purpose tool is used to generate *verification conditions* (*VC*s)

This tool, called a *verification condition generator* (*VCG*), contains the language semantics, e.g., encoded as Hoare-triples.

Most practical VCGs do a lot of simplification as they build VCs.

A theorem prover is used to prove the VCs.


labels
program π
paths
assertions
α
β
γ
RETURN
f(s)
t
g(s)
h(s)
P(s)
pre−condition
R(s)
loop invariant
Q(s)
post−conditior

We assume the program in s, π, does not change during execution.

We assume there is exactly one `HALT`, at γ.

Let s_0 be the initial state of program π.

$pc\,(s_0) = \alpha$

Let s_k denote $\mathit{m1}\,(s_0, k)$.

Partial Correctness:
$P\,(s_0)\ \wedge\ \mathit{haltedp}\,(s_k) \rightarrow Q(s_k)$.

We assume the program in s, π, does not change during execution.

We assume there is exactly one `HALT`, at γ.

Let s_0 be the initial state of program π.

$$pc\,(s_0) = \alpha$$

Let s_k denote $m1\,(s_0, k)$.

Partial Correctness:
$P\,(s_0)\ \wedge\ haltedp\,(s_k) \rightarrow Q(s_k)$.

We assume the program in s, π, does not change during execution.

We assume there is exactly one `HALT`, at γ.

Let s_0 be the initial state of program π.

$pc\,(s_0) = \alpha$

Let s_k denote $m1\,(s_0, k)$.

Partial Correctness:
$P\,(s_0)\ \wedge\ pc\,(s_k) = \gamma \rightarrow Q(s_k)$.

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

(Actually, we assert "$prog(s) = \pi$" at α, β and γ, but we omit that here by our convention that the program is always π.)

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \to Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

Objection: Is this definition consistent? Yes: Every tail-recursive definition is witnessed by a total function [Manolios and Moore, 2000]. See ACL2 Community Books `misc/defpun` and `misc/defp`.

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

Assume we've proved

$$Inv(s) \rightarrow Inv(step(s)).$$

(We'll see the proof in a moment.)

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

$Inv(s_0) \rightarrow Inv(s_k)$ *(By induction)*

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

$$Inv(s_0) \rightarrow Inv(s_k)$$

$$pc(s_0) = \alpha \qquad \textit{(By def of } s_0\textit{)}$$

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s_0) \equiv \begin{cases} P(s_0) & \text{if } pc(s_0) = \alpha \\ R(s_0) & \text{if } pc(s_0) = \beta \\ Q(s_0) & \text{if } pc(s_0) = \gamma \\ Inv(step(s_0)) & \text{otherwise} \end{cases}$$

$$Inv(s_0) \rightarrow Inv(s_k)$$

$$pc(s_0) = \alpha \qquad \textit{(By def of } s_0\textit{)}$$

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

$$P(s_0) \rightarrow Inv(s_k)$$

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

$P(s_0) \rightarrow Inv(s_k)$

$P(s_0) \qquad (Given)$

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

$Inv(s_k)$

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

$Inv(s_k)$

$pc(s_k) = \gamma \qquad (Given)$

Theorem: $P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s_k) \equiv \begin{cases} P(s_k) & \text{if } pc(s_k) = \alpha \\ R(s_k) & \text{if } pc(s_k) = \beta \\ Q(s_k) & \text{if } pc(s_k) = \gamma \\ Inv(step(s_k)) & \text{otherwise} \end{cases}$$

$Inv(s_k)$

$pc(s_k) = \gamma$ $\qquad$ $(Given)$

Theorem: $P(s_0) \wedge pc(s_k) = \gamma \to Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

$Q(s_k)$

Q.E.D.

So it's trivial to prove the theorem

$$P\,(s_0) \wedge pc\,(s_k) = \gamma \rightarrow Q\,(s_k)$$

if we can prove

$$Inv\,(s) \rightarrow Inv\,(step\,(s)).$$

$$Inv\,(s) \equiv \begin{cases} P\,(s) & \text{if } pc\,(s) = \alpha \\ R\,(s) & \text{if } pc\,(s) = \beta \\ Q\,(s) & \text{if } pc\,(s) = \gamma \\ Inv\,(step\,(s)) & \text{otherwise} \end{cases}$$

$Inv(s) \rightarrow Inv(step(s))$

Proof.

Expanding $Inv(s)$ generates four cases:

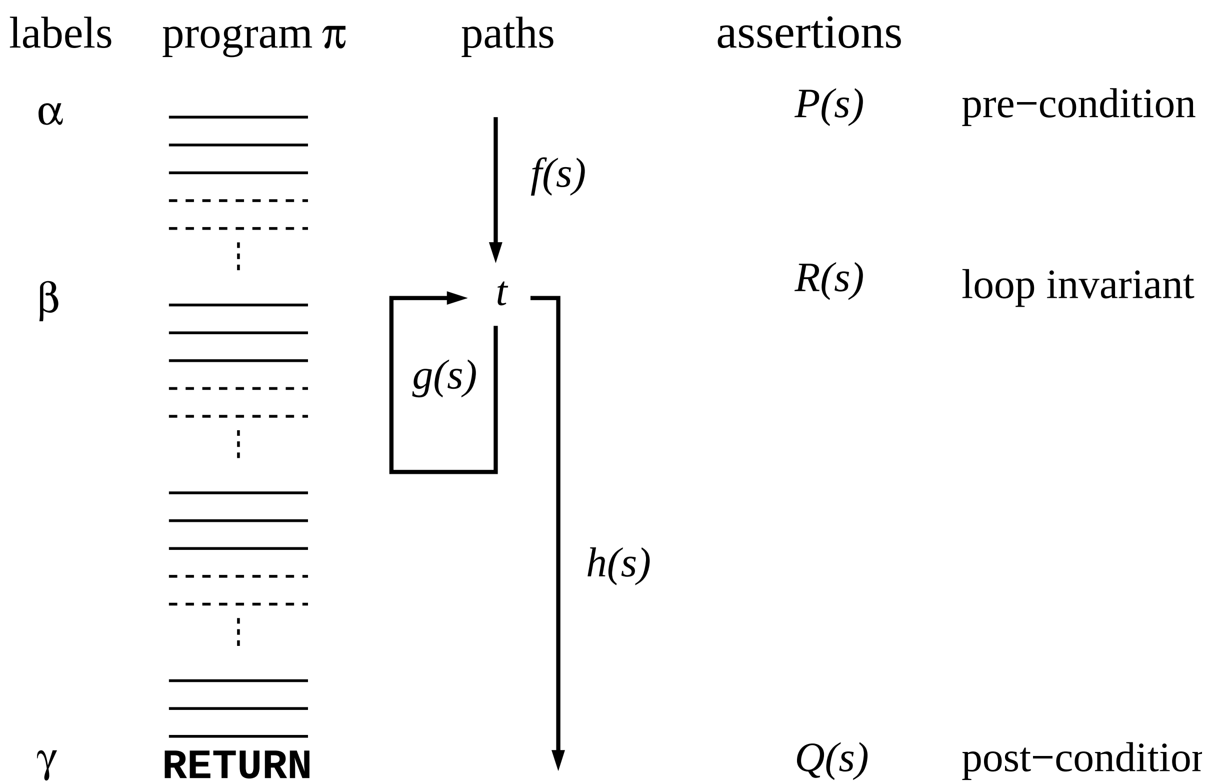
Case $pc(s) = \alpha$:
Case $pc(s) = \beta$:
Case $pc(s) = \gamma$:
Case *otherwise*:

$$Inv\,(s) \equiv \begin{cases} P\,(s) & \text{if } pc\,(s) = \alpha \\ R\,(s) & \text{if } pc\,(s) = \beta \\ Q\,(s) & \text{if } pc\,(s) = \gamma \\ Inv\,(step\,(s)) & \text{otherwise} \end{cases}$$

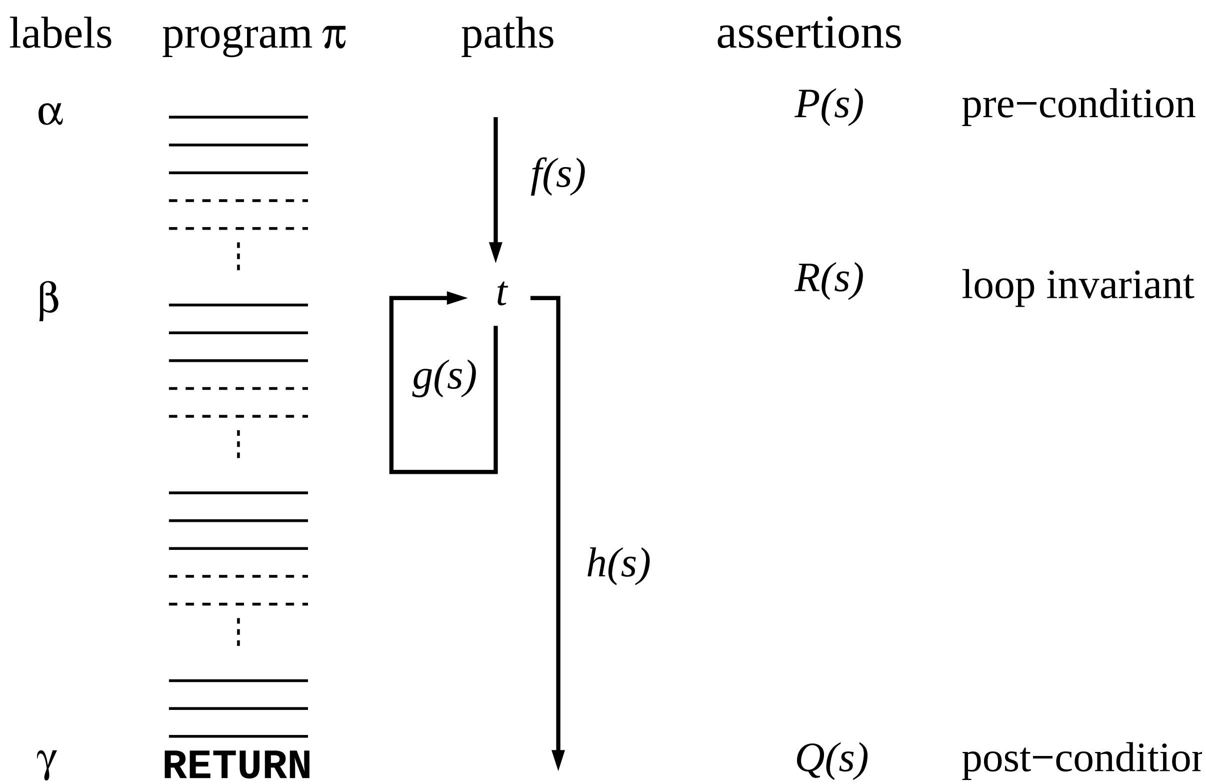
$$Inv\,(s) = Inv\,(step\,(s)) = Inv\,(step\,(step\,(s)))\ldots$$

as long as the $pc \notin \{\alpha, \beta, \gamma\}$.

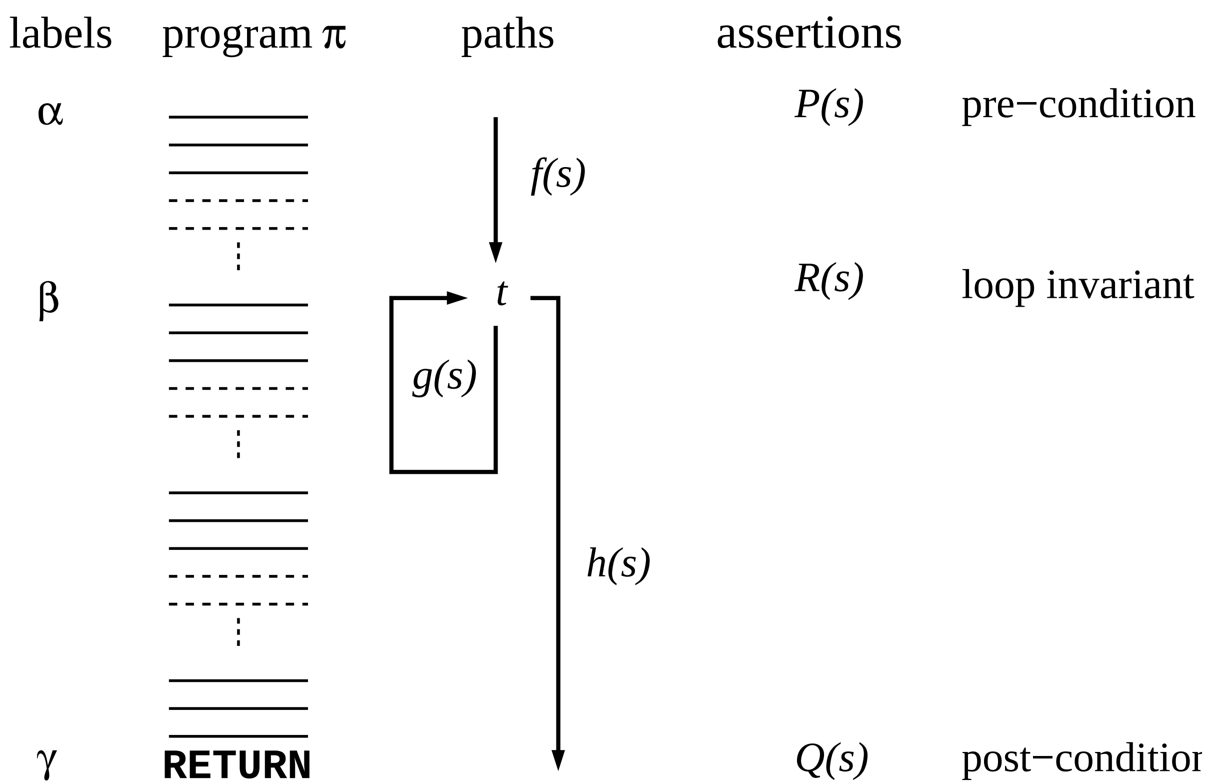
$$Inv\,(s) \rightarrow Inv\,(step\,(s)) \qquad \text{[Case } pc(s) = \alpha\text{]}$$



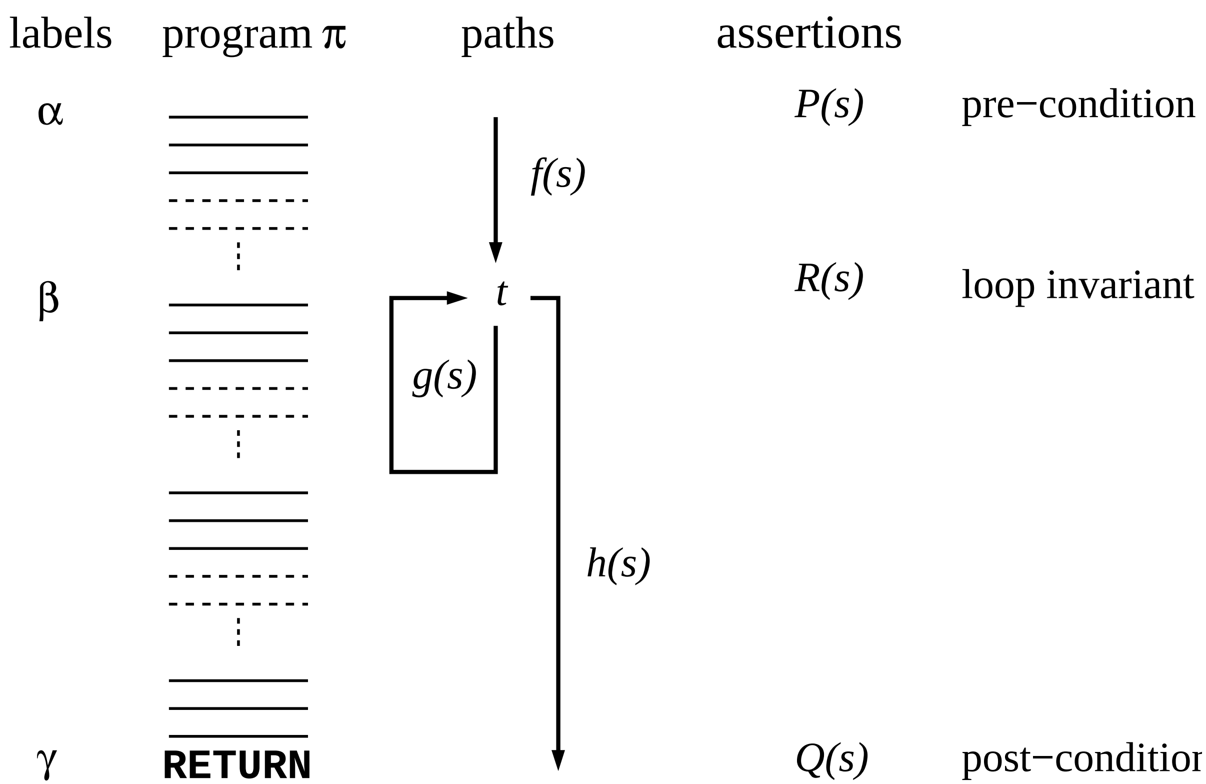
$$P(s) \rightarrow Inv(step(s)) \qquad \text{[Case } pc(s) = \alpha\text{]}$$



$P(s) \to R(f(s))$ [Case $pc(s) = \alpha$]

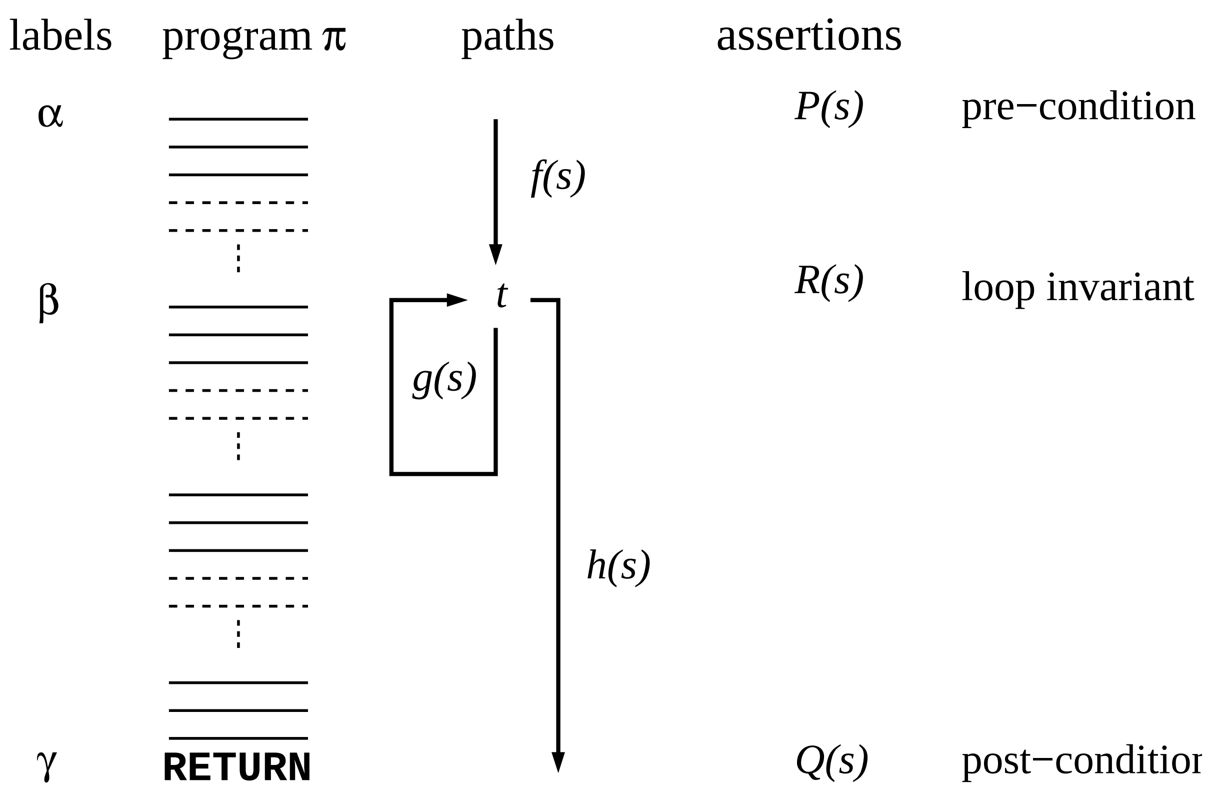


$$Inv\,(s) \rightarrow Inv\,(step\,(s)) \qquad \text{[Case } pc(s) = \beta\text{]}$$

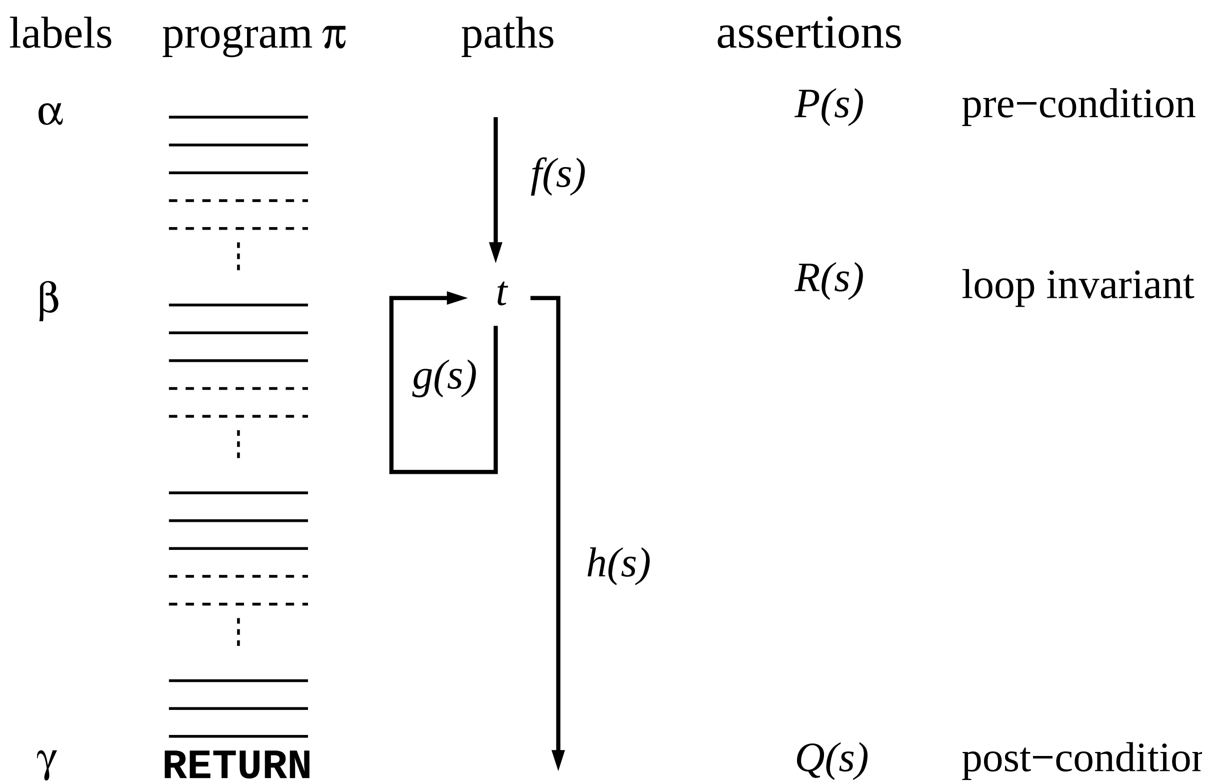


$$(R(s) \wedge t \to R(g(s)))$$
$$(R(s) \wedge \neg t \to Q(h(s)))$$

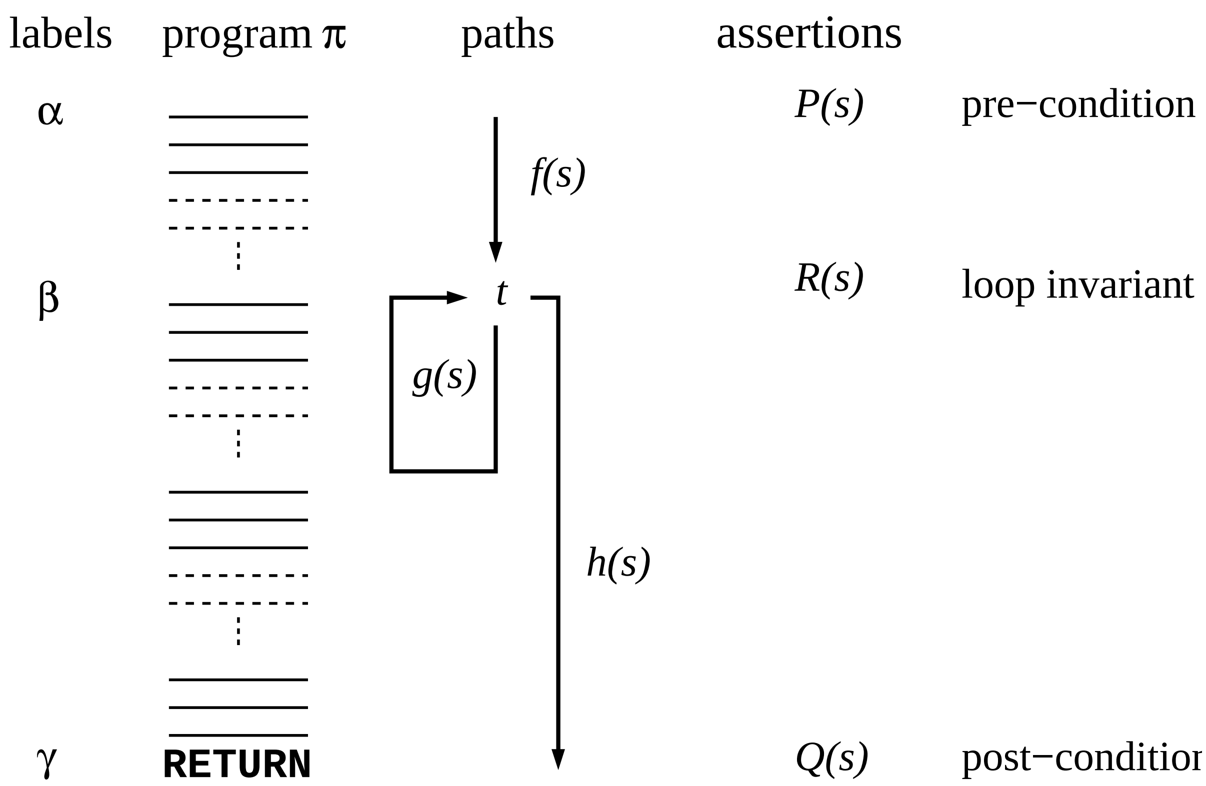
[Case $pc(s) = \beta$]



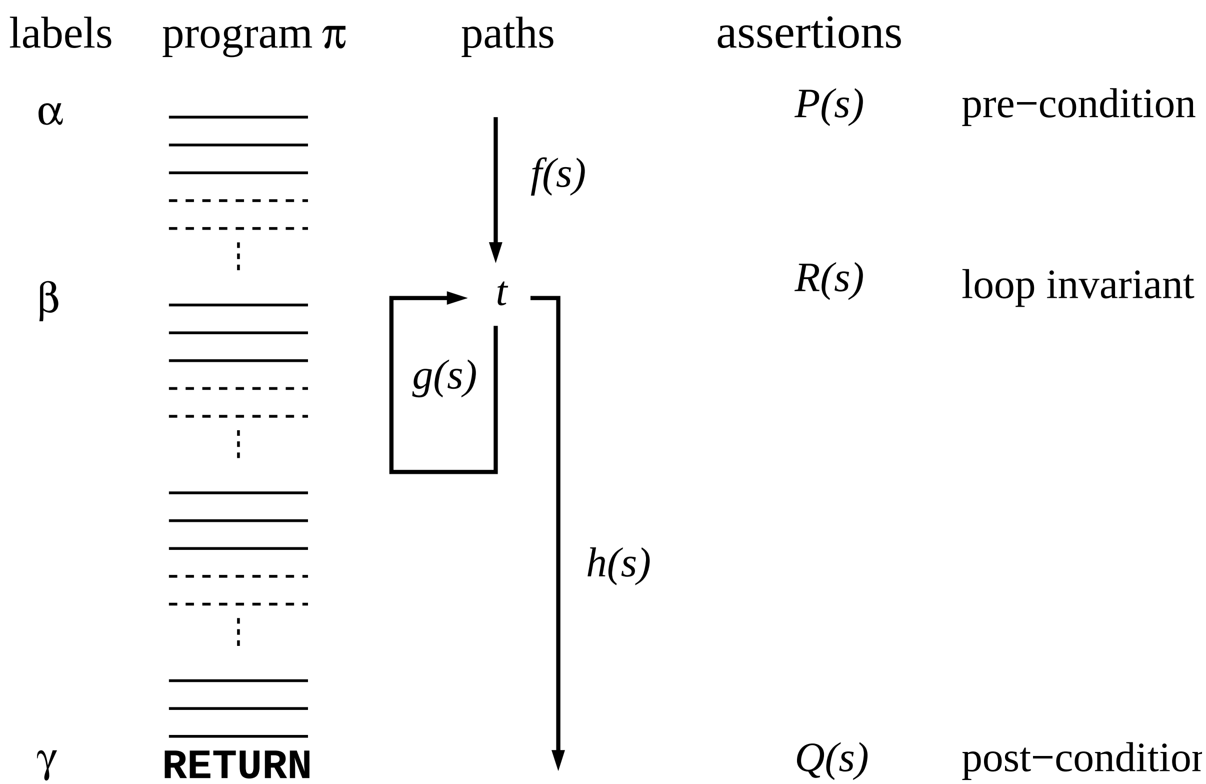
$$Inv\,(s) \rightarrow Inv\,(step\,(s)) \qquad \text{[Case } pc(s) = \gamma\text{]}$$



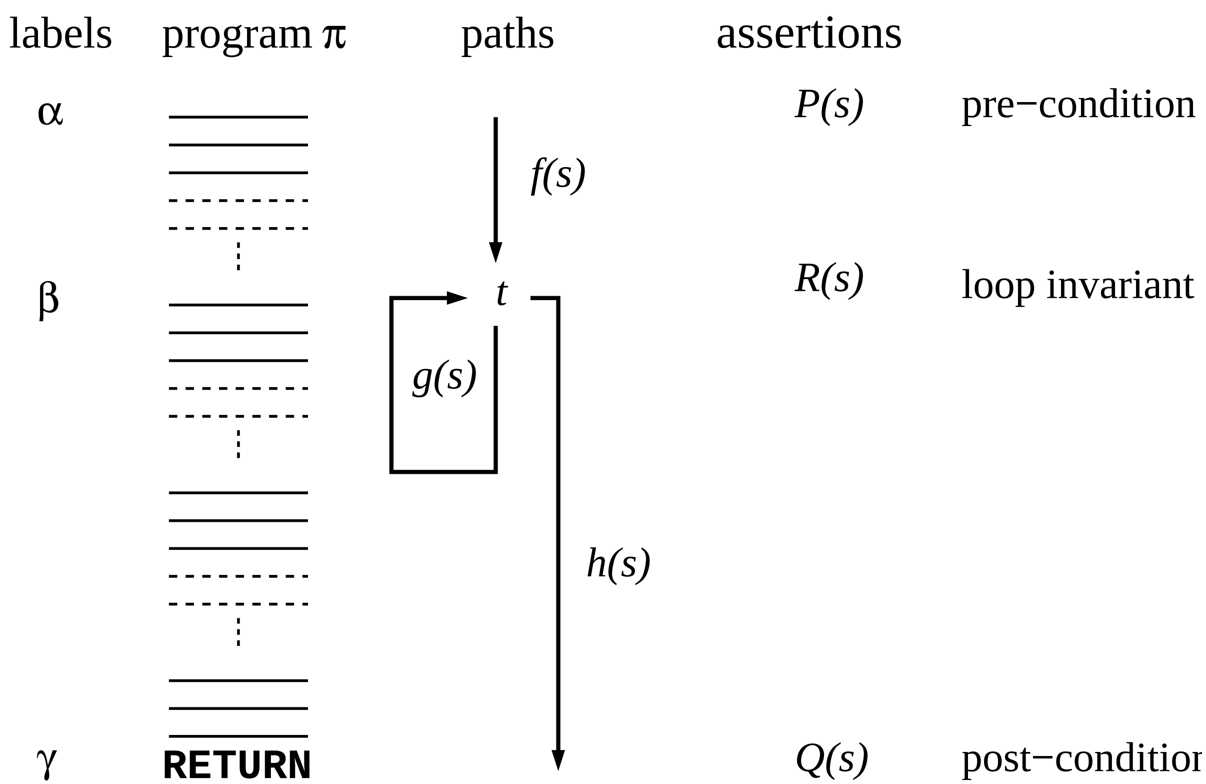
$$Inv\,(s) \rightarrow Inv\,(s) \qquad \text{[Case } pc(s) = \gamma\text{]}$$



$Inv\,(s) \rightarrow Inv\,(step\,(s))$ [Case *otherwise*]



$Inv\,(step\,(s)) \rightarrow Inv\,(step\,(s))$ [Case *otherwise*]



Recap: Given the definition of Inv , the "natural" proof of

$Inv\,(s) \rightarrow Inv\,(step\,(s))$

generates the standard verification conditions

VC1. $P\,(s) \rightarrow R\,(f\,(s))$,
VC2. $R\,(s) \wedge t \rightarrow R\,(g\,(s))$, and
VC3. $R\,(s) \wedge \neg t \rightarrow Q\,(h(s))$
as subgoals from the operational semantics!

It generates no other non-trivial proof obligations.

The VCs are simplified as they are generated.

Demo

Discussion

We did not write a VCG for M1.

The VCs were generated directly from the operational semantics by the theorem prover.

Since VCs are generated by proof, the paths explored and the VCs generated are sensitive to the pre-condition specified.

The VCs are simplified (and possibly proved) by the same process.

We did not count instructions or define a clock function.

We did not constrain the inputs so that the program terminated.

Indeed, we can deal with non-terminating programs.

Demo

Total Correctness via Inductive Assertions

We have also handled total correctness via the VCG approach.

An ordinal measure is provided at each cut point and the VCs establish that it decreases upon each arrival at the cut point.

Clock functions can be automatically generated and admitted from such proofs.

See Sandip Ray's `proofstyles/` books in the Community Books.

Inductive Assertion Tools

See the following Community Books:

`symbolic/generic/defsimulate.lisp`

`workshops/2011/krug-et-al/support/Symbolic/`
    `defsimulate+.lisp`

See also paper
`http://www.cs.utexas.edu/users/sandip/`
    `publications/symbolic/main.html`
which describes the `defsimulate` book.

Citations

P. Manolios and J Moore, "Partial Functions in ACL2," *JAR* 2003.

J Moore, "Inductive Assertions and Operational Semantics," *CHARME 2003*, D. Geist (Ed.), Springer Verlag LNCS 2860, pp. 289–303, 2003.

J Moore, **S. Ray**, W. Hunt, and J. Matthews, "A Mechanical Analysis of Program Verification Strategies," *Journal of Automated Reasoning*, **40**(4), pp. 245–269, May 2008.

Additional Material

J Moore, "Proving Theorems about Java and the JVM with ACL2", *Models, Algebras and Logic of Engineering Software*, M. Broy and M. Pizka (eds), IOS Press, Amsterdam, pp 227-290, 2003.

`http://www.cs.utexas.edu/users/moore/publications/marktoberdorf-02/main.pdf`

and also ACL2 Community Books `models/jvm/m5/` for examples of subroutine call/return, heap manipulation, and multi-threading.

Subroutine Call Tips

- define `(poised-to-invoke-`*name* `s)`
- if using clocks, define `(clk-`*name* `s)` to count instructions from call through return
- specify correctness to include restoration of (relevant) registers and stack, preservation of the program segment, and advancement of pc to the next instruction

- use `(whatever-it-is-... s ...)` wormhole abstraction to specify “don’t care” values for those parts of the state that you *will never* care about

- specify correct answer

Multi-Threading Tip

Try to reduce it to sequential correctness in a slightly chaotic environment

J Moore, "A Mechanically Checked Proof of a Multiprocessor Result via a Uniprocessor View," *Formal Methods in System Design*, **14**(2), March, 1999, pp. 213-228.

`http://www.cs.utexas.edu/users/moore/publications/multi-v-uni.pdf`