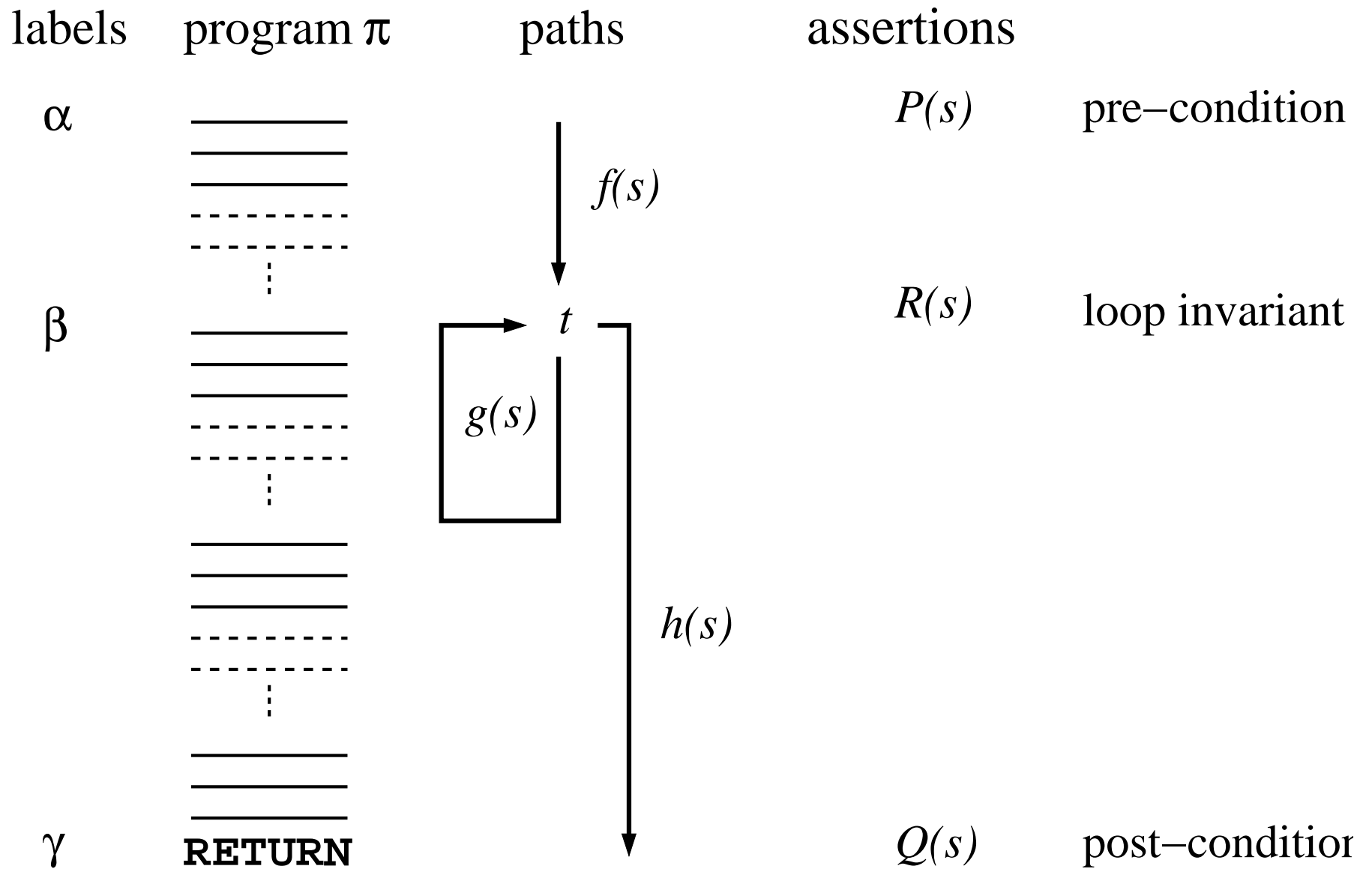


Mechanized Operational Semantics

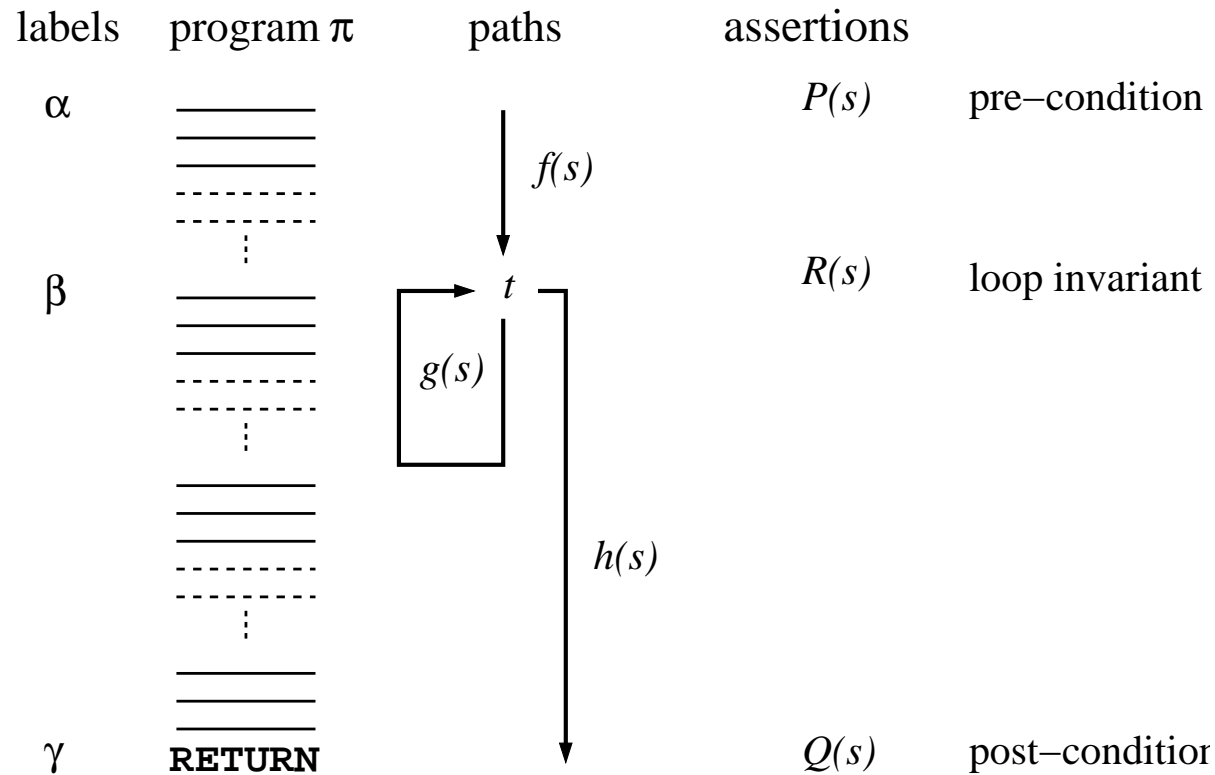
J Strother Moore
Department of Computer Sciences
University of Texas at Austin

Marktoberdorf Summer School 2008

(Lecture 4: Inductive Invariant Proofs)

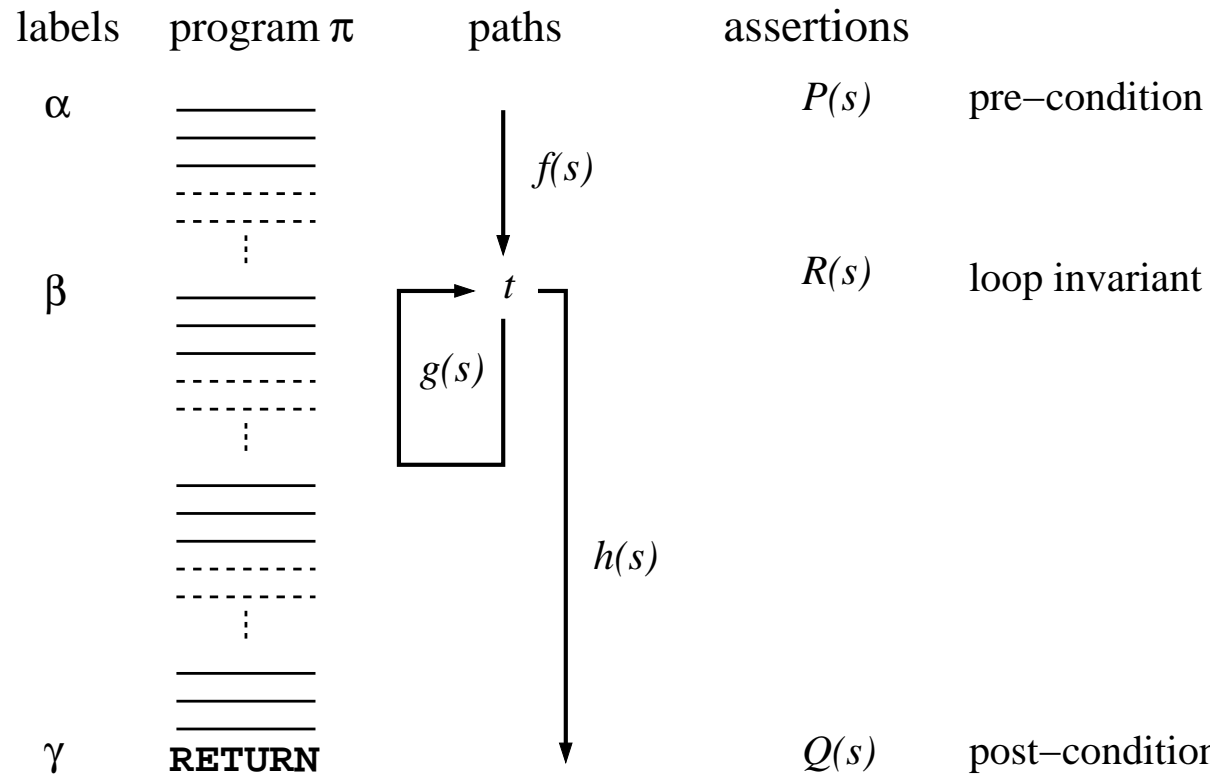


Conventional Mechanized Code Proofs



$$\text{VC1. } P(s) \longrightarrow R(f(s)),$$

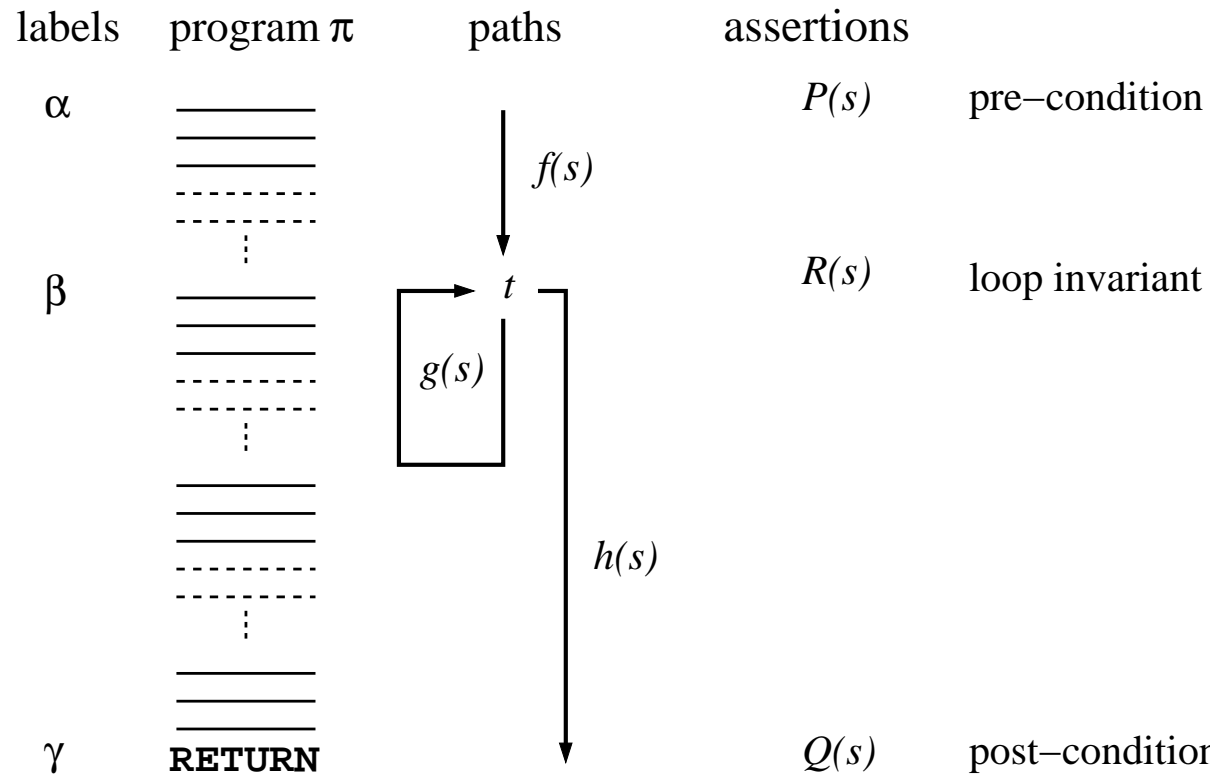
Conventional Mechanized Code Proofs



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

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

Conventional Mechanized Code Proofs



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

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

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

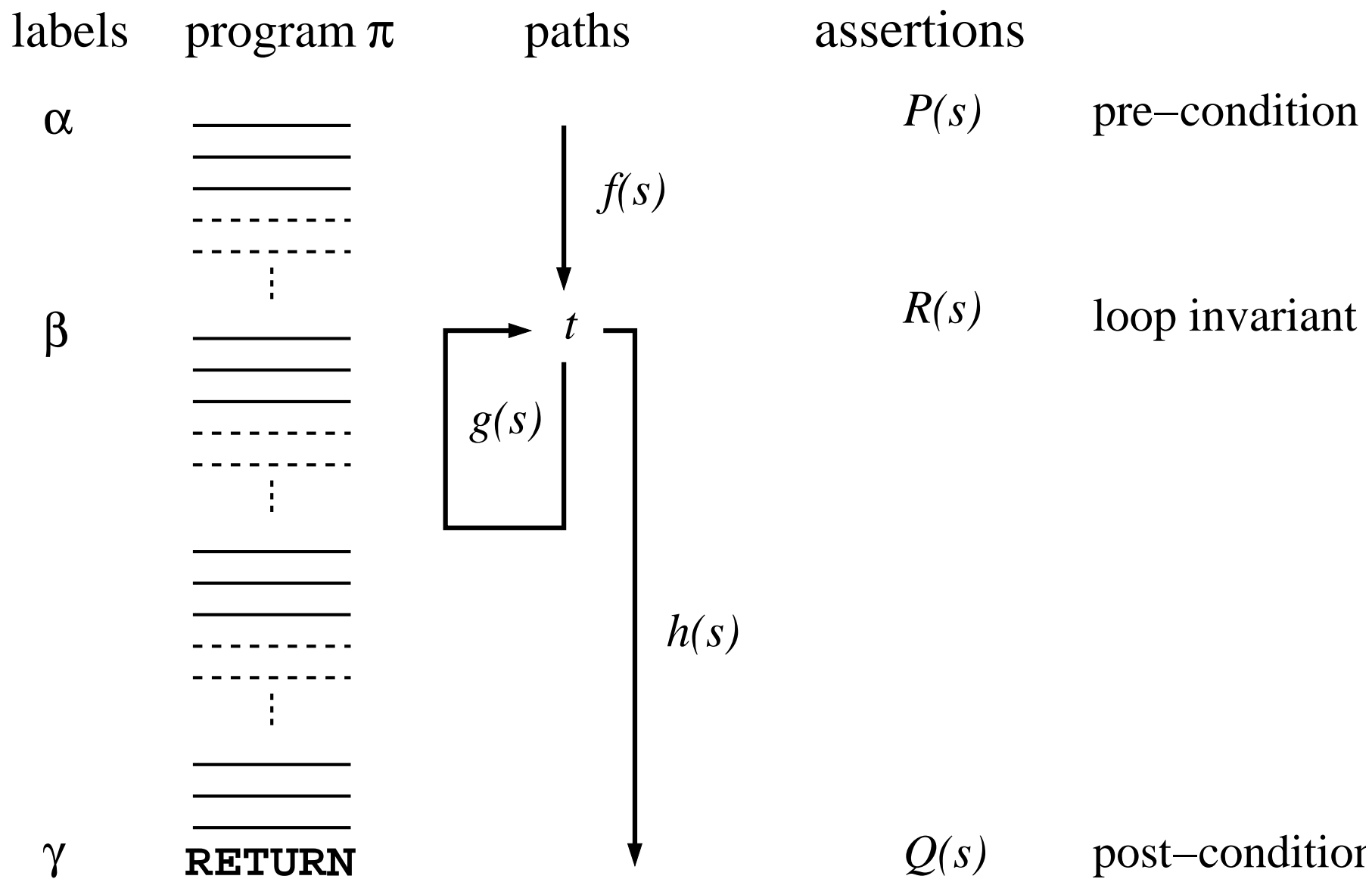
Conventional Mechanized Code Proofs

The process by which proof obligations (*verification conditions* or *VCs*) are generated from the code is called *verification condition generation* and is performed by a *VCG* program.

Typically, VCGs simplify the VC “on-the-fly.”

Typically, the language semantics is coded into the VCG.

A separate theorem prover is used to prove the VCs.



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

Let s_0 be the initial state of program π .

$$pc(s_0) = \alpha$$

Let s_k denote $run(k, s_0)$.

Formally Stated Correctness Theorems

Total:

$$\exists k : P(s_0) \rightarrow (Q(s_k) \wedge pc(s_k) = \gamma).$$

$$\exists k : P(s_0) \rightarrow (Q(run(k, s_0)) \wedge \dots)$$

This is sometimes stated without the quantifier as

$$P(s_0) \rightarrow (Q(run(sched(s_0), s_0)) \wedge \dots).$$

Partial:

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

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

Proof: Define

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

Objection: Is this definition consistent? Yes: Every tail-recursive definition is witnessed by a total function. (Manolios and Moore, 2000)

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

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

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

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

Proof: Define

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

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

$$pc(s_0) = \alpha \quad (By\ construction)$$

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

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

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

$$P(s_0) \quad (Given)$$

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

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

$Inv(s_k)$

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

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

Proof: Define

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

$Q(s_k)$

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 $pc(s) = \alpha$:

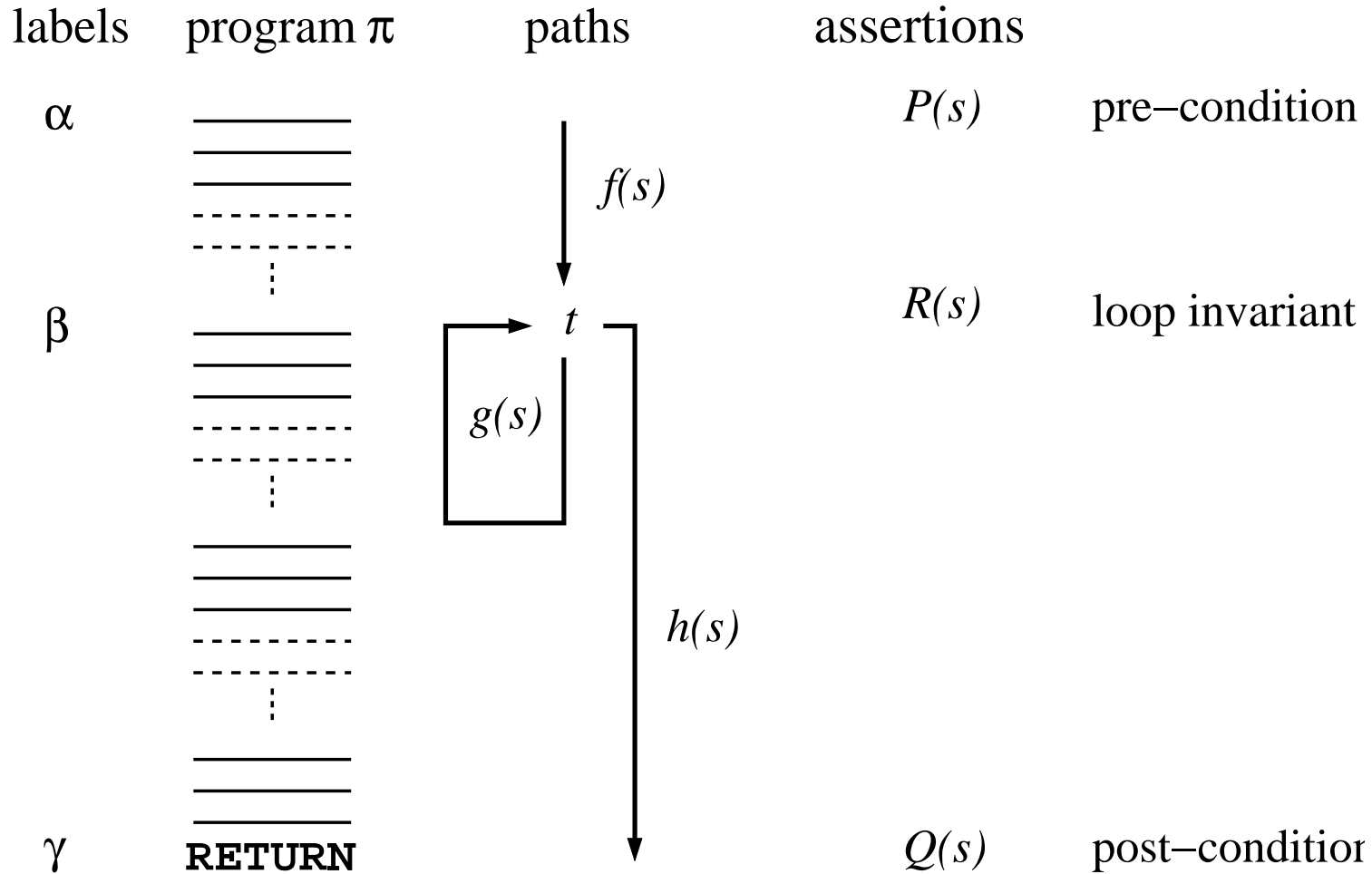
Case $pc(s) = \beta$:

Case $pc(s) = \gamma$:

Case *otherwise*:

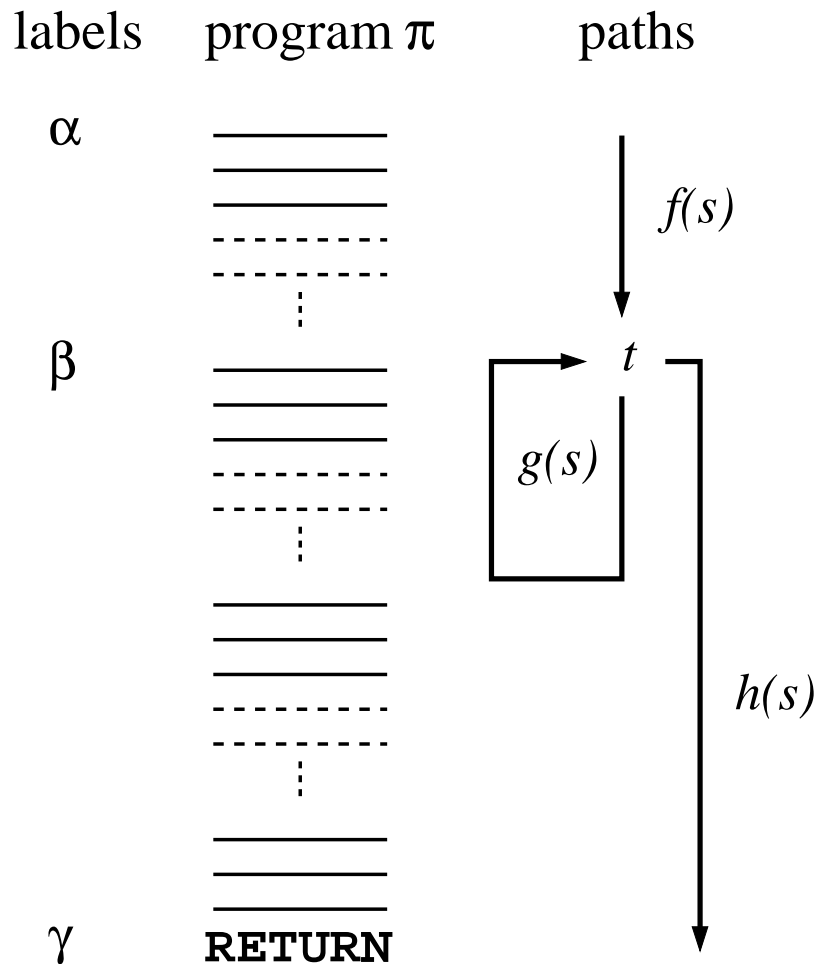
$$Inv(s) \rightarrow Inv(step(s))$$

$$[\text{Case } pc(s) = \alpha]$$



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

$$[\text{Case } pc(s) = \alpha]$$



assertions

$P(s)$ pre-condition

$R(s)$ loop invariant

$Q(s)$ post-condition

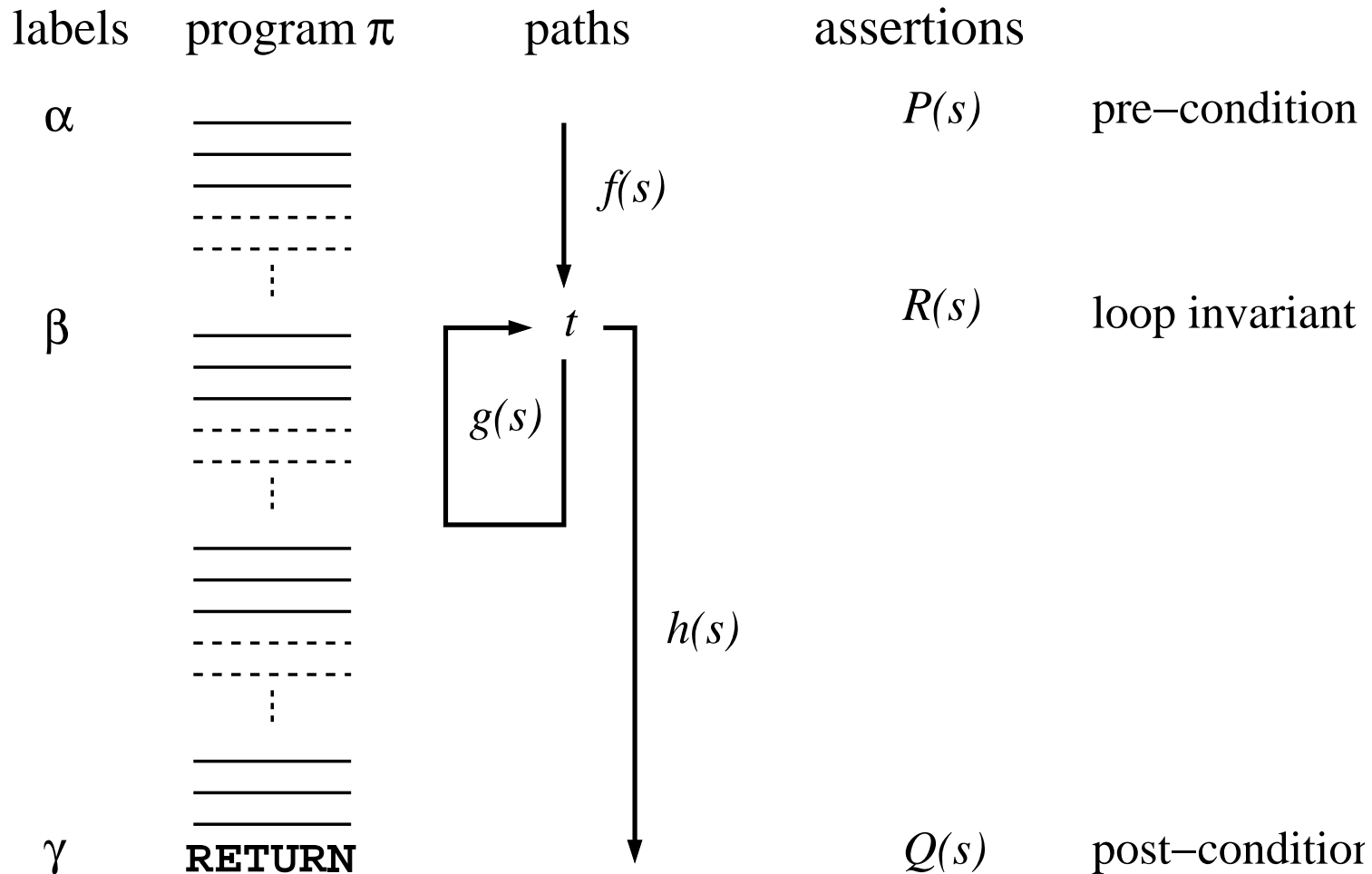
$$Inv(s) \equiv \begin{cases} P(s) & \text{if } pc(s) = \alpha \\ R(s) & \text{if } pc(s) = \beta \\ Q(s) & \text{if } pc(s) = \gamma \\ Inv(step(s)) & \text{otherwise} \end{cases}$$

$$Inv(s) = Inv(step(s)) = Inv(step(step(s))) \dots$$

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

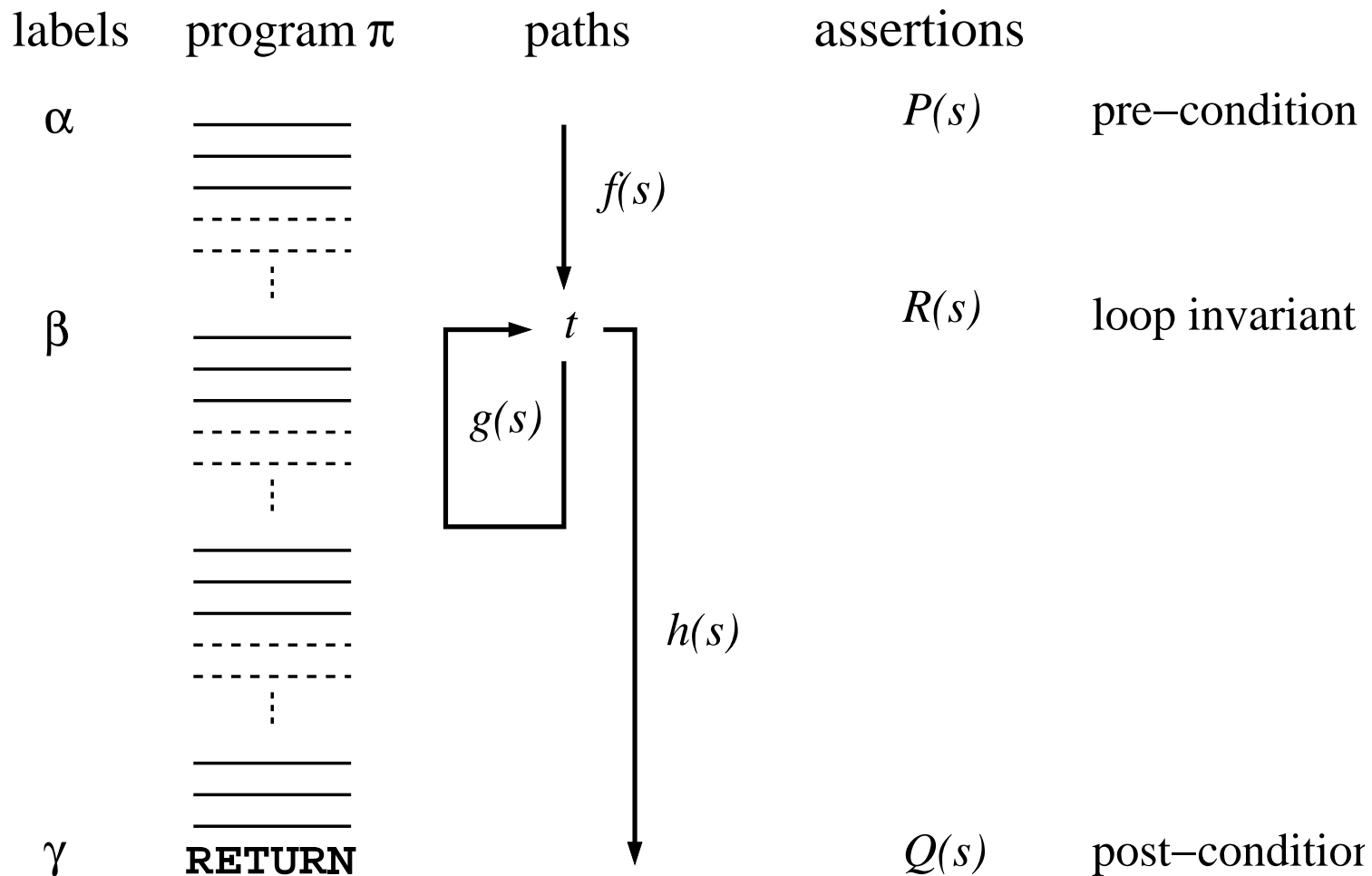
$$P(s) \rightarrow Inv(step(s))$$

$$[\text{Case } pc(s) = \alpha]$$



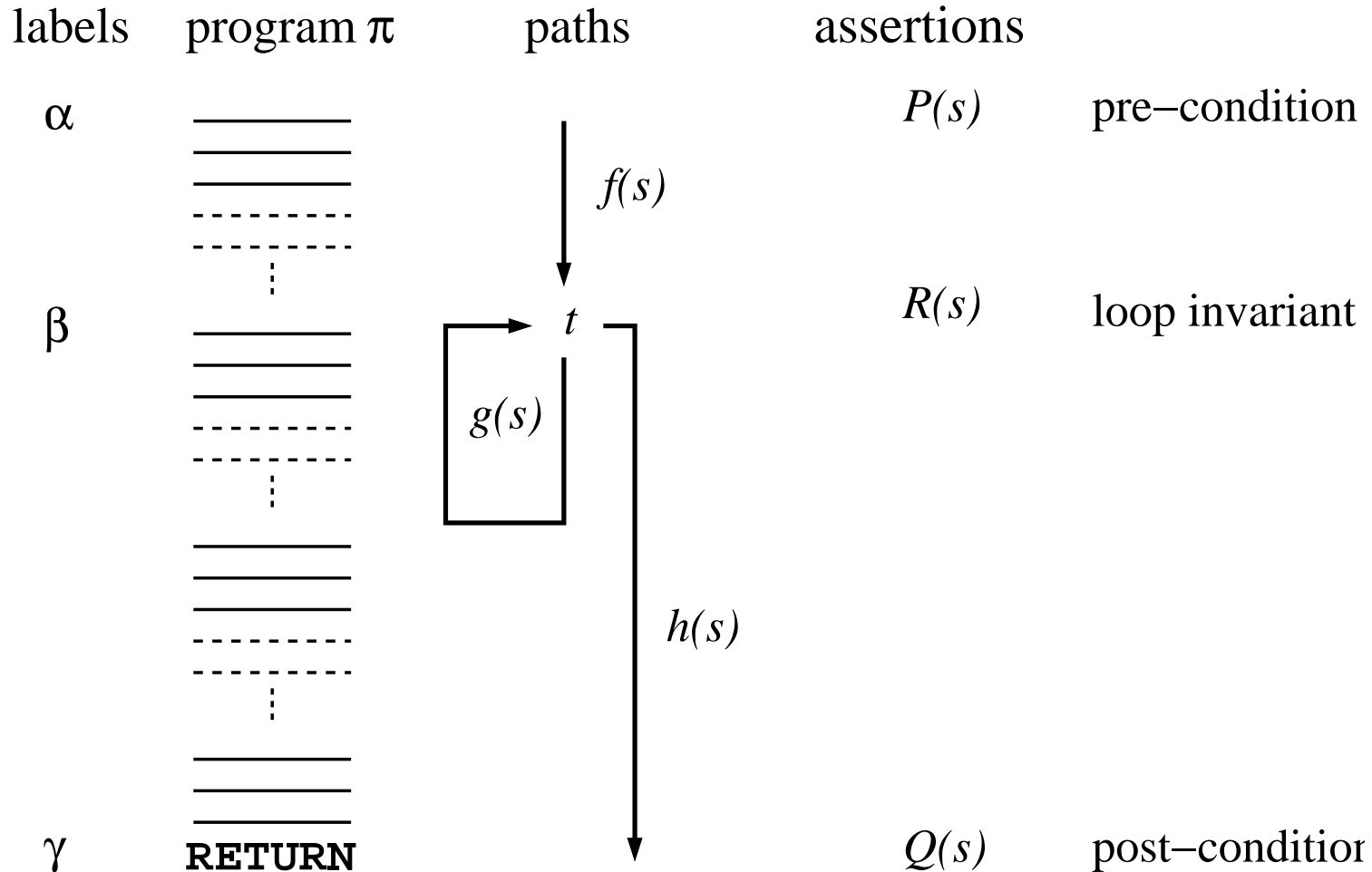
$$P(s) \rightarrow R(f(s))$$

$$[\text{Case } pc(s) = \alpha]$$



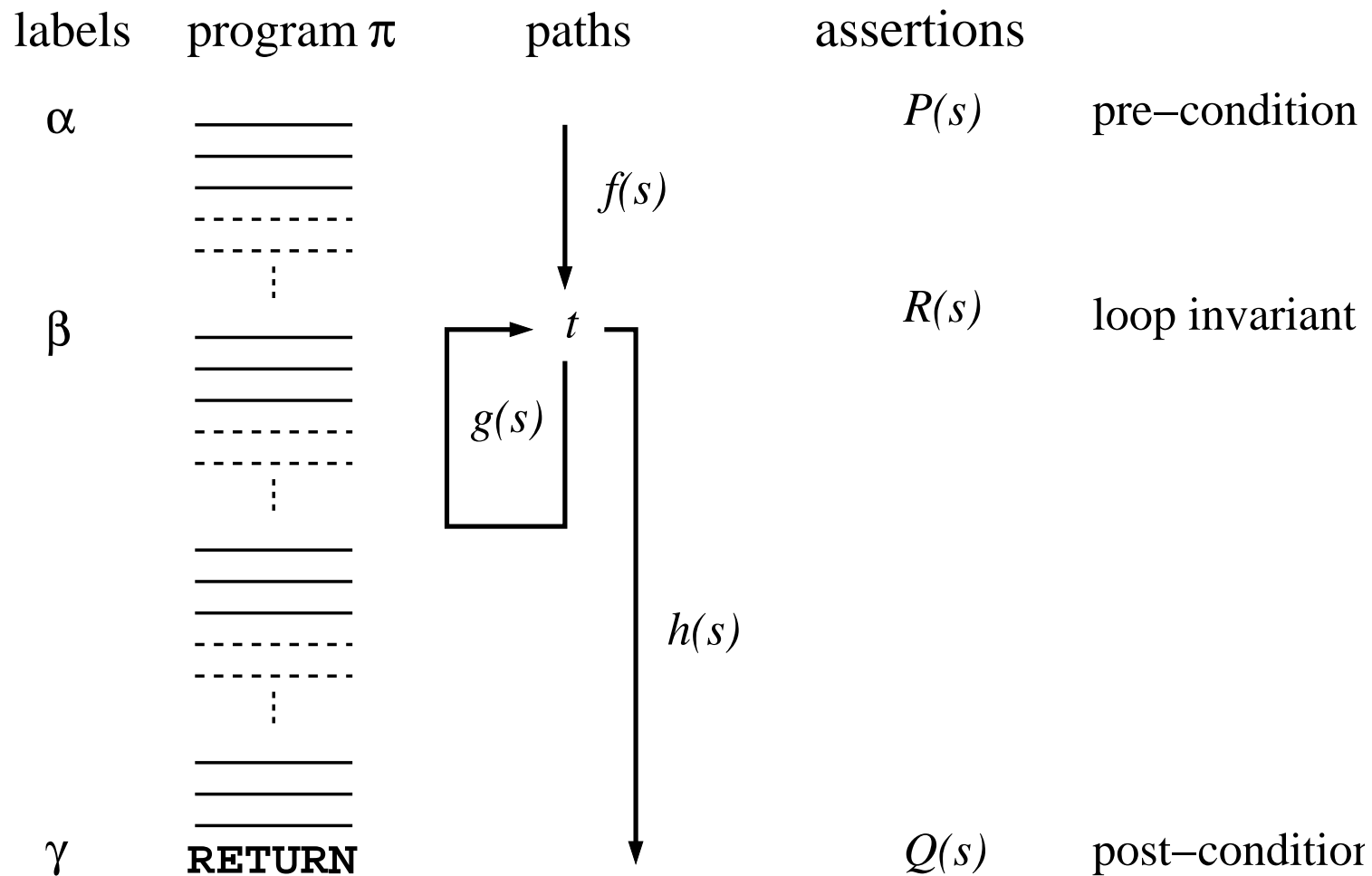
$$Inv(s) \rightarrow Inv(step(s))$$

$$[\text{Case } pc(s) = \beta]$$



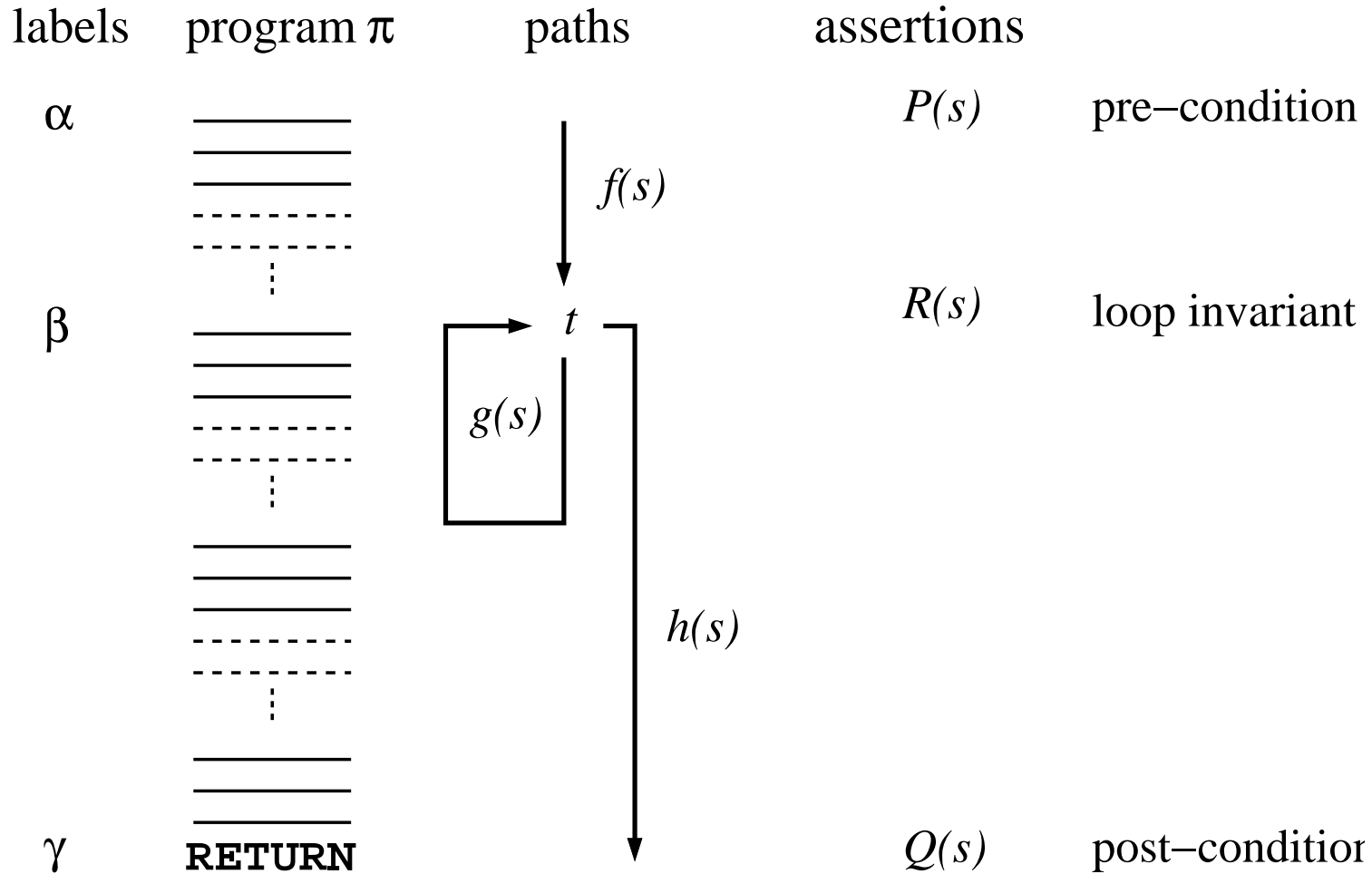
$$\begin{aligned}
 & (R(s) \wedge t \rightarrow R(g(s))) \\
 & (R(s) \wedge \neg t \rightarrow Q(h(s)))
 \end{aligned}$$

$$[\text{Case } pc(s) = \beta]$$



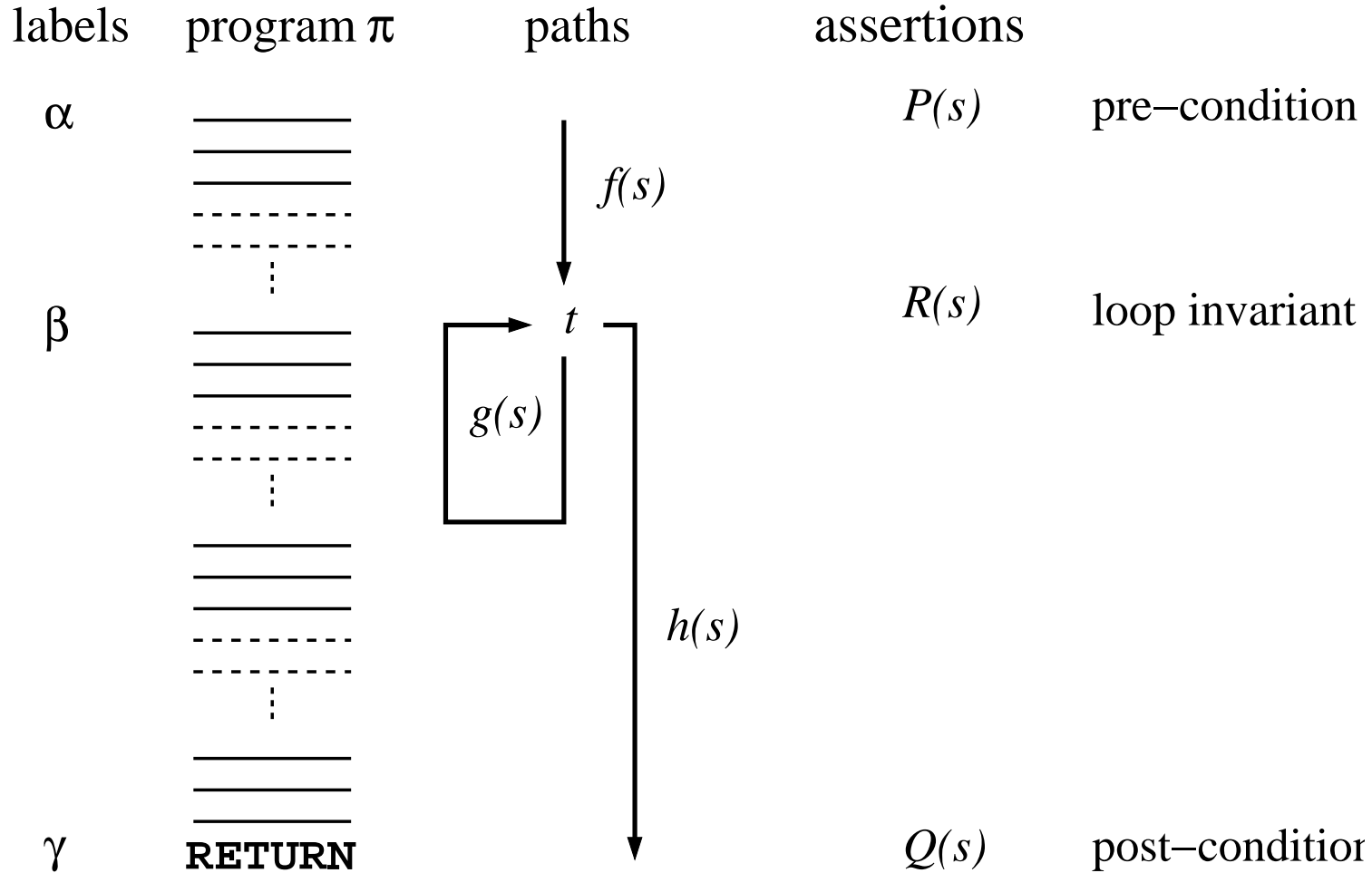
$$Inv(s) \rightarrow Inv(step(s))$$

$$[\text{Case } pc(s) = \gamma]$$



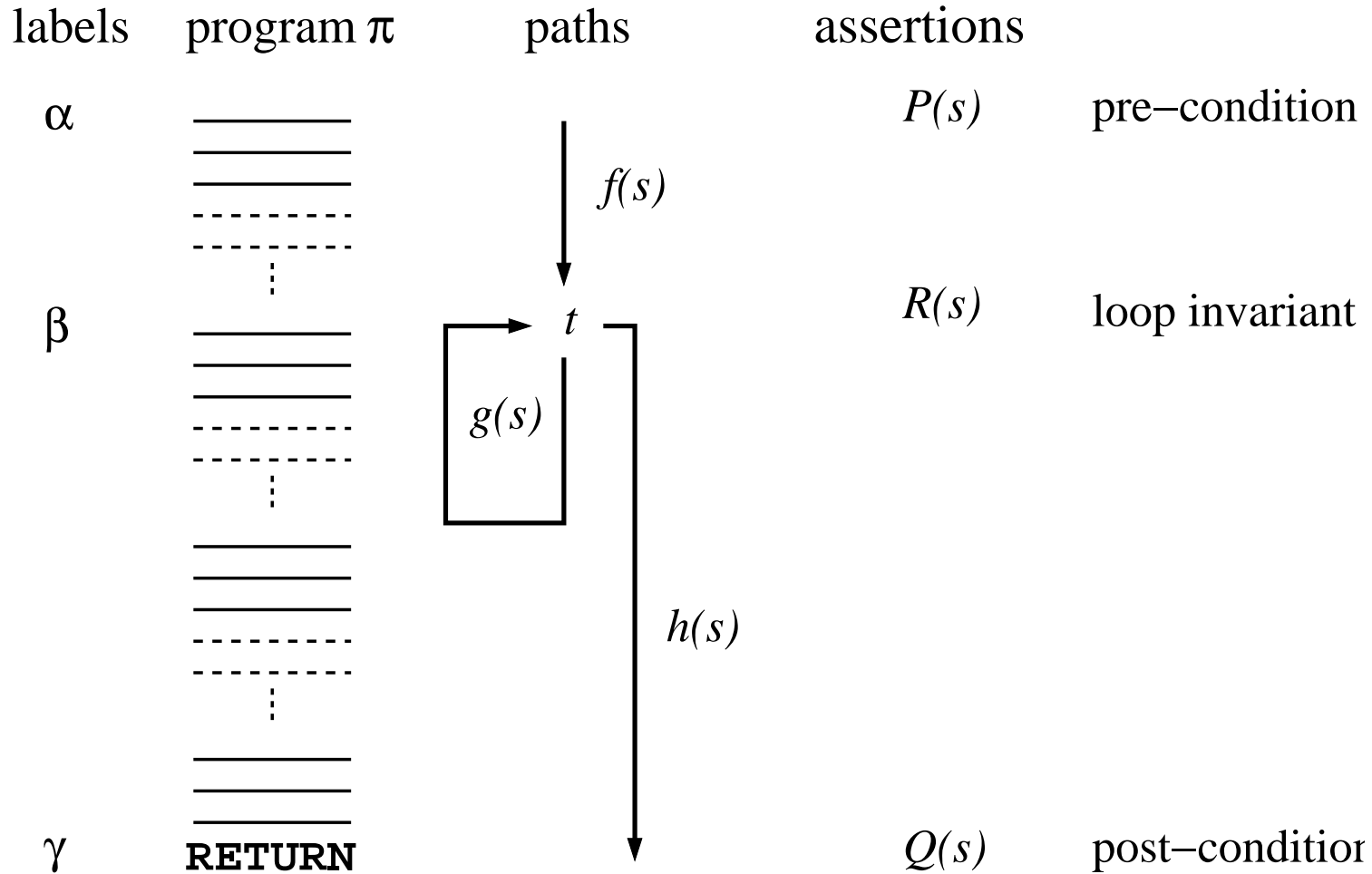
$$Inv(s) \rightarrow Inv(s)$$

$$[\text{Case } pc(s) = \gamma]$$

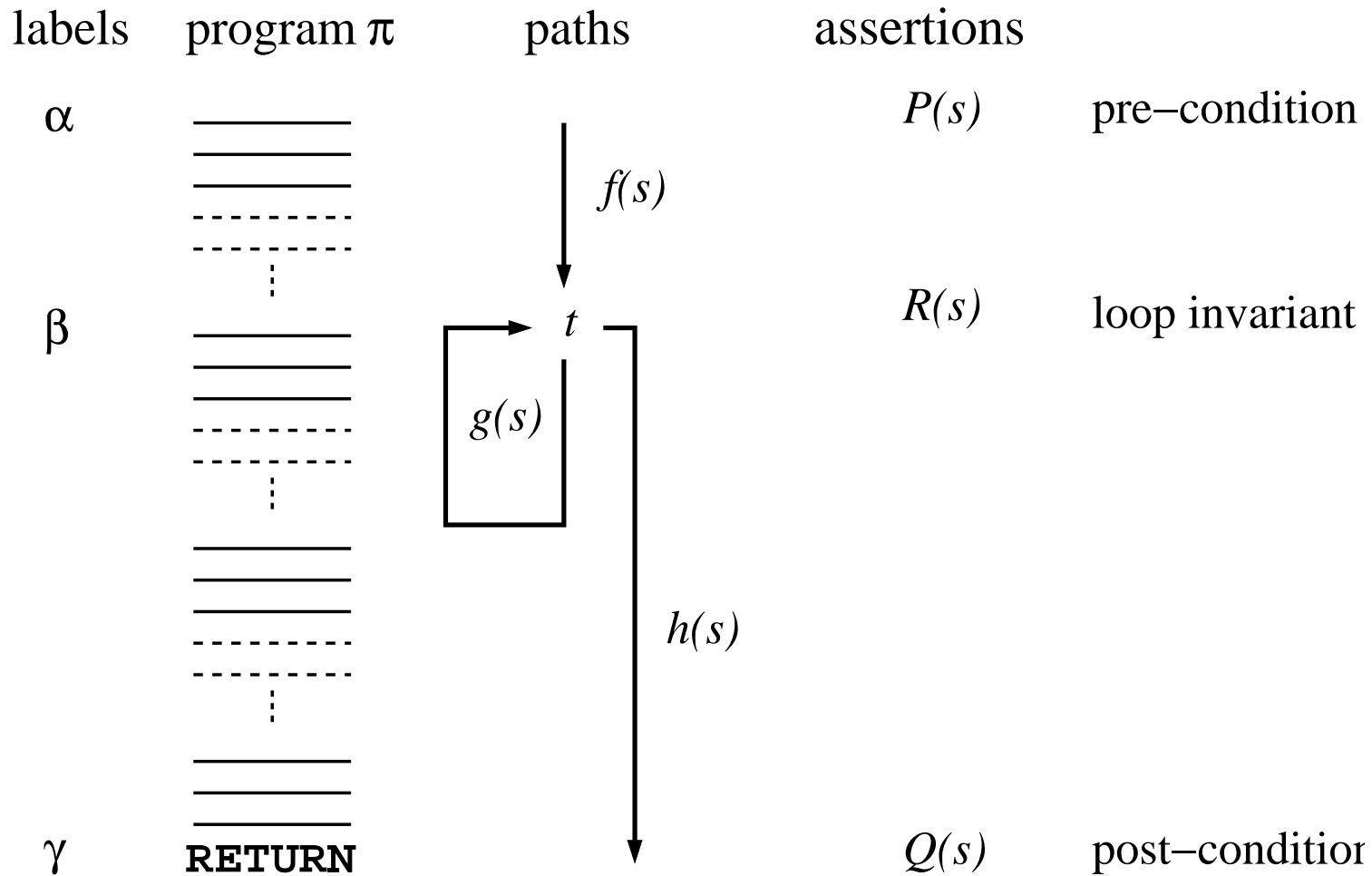


$$Inv(s) \rightarrow Inv(step(s))$$

[Case *otherwise*]



$$Inv(step(s)) \rightarrow Inv(step(s)) \text{ [Case otherwise]}$$



Recap: Given the definition of Inv , the “natural” proof of

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

generates the standard verification conditions

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

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

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

as subgoals from the operational semantics!

It generates no other non-trivial proof obligations.

The VCs are simplified as they are generated.

Demo 1

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 schedule.

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

Indeed, we can deal with non-terminating programs.

Demo 2

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.

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

Primary Citation

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

Other Examples

Nested loops are handled exactly as by standard VCG methods.

```
public static int tfact(int n){ /* Factorial by repeated addition.      */
    int i = 1;                  /* Verified using inductive assertions */
    int b = 1;                  /* by Alan Turing, 1949.              */
    while (i<=n){
        int j = 1;
        int a = b;
        while (j < i) {
            b = a+b;
            j++;
        };
        i++;
    };
    return b;
}
```

Recursive methods can be handled.

```
public static int fact(int n){  
    if (n>0)  
        {return n*fact(n-1);}  
    else return 1;  
}
```

To handle recursive methods we

- modify *run* to terminate upon top-level return, and
- add a standard invariant about the shape of the call stack.

Conclusion

If you have

- a theorem prover and
- a formal operational semantics,

you can prove formally stated *partial program correctness* theorems using *inductive assertions* without building or verifying a VCG.

Related Work

P. Y. Gloess, “Imperative Program Verification in PVS,” École Nationale Supérieure Électronique, Informatique et Radiocommunications de Bordeaux, 1999.

P. Homeier and D. Martin, “A Mechanically Verified Verification Condition Generator,” *The Computer Journal*, **38**(2), pp. 131–141, July 1995.

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

J. Matthews, J S. Moore, S. Ray, and D. Vroon:
“Verification Condition Generation via Theorem
Proving,” to appear in M. Hermann and A.
Voronkov, editors, *Proceedings of the 13th
International Conference on Logic for Programming,
Artificial Intelligence, and Reasoning (LPAR 2006)*,
Phnom Penh, Cambodia, November 2006,
Springer-Verlag.

Next Time

a much more interesting correctness proof