A Direct Soundness Proof of the Recursion Rule

by

N. Francez and O. Grumberg

Technical Report #684
July 1991
A Direct Soundness Proof of the Recursion Rule

Nissim Francez and Orna Grumberg
Computer Science Department, The Technion
Haifa 32000, Israel
E-address: francez@tecel.bitnet, francez@cs.technion.ac.il
Phone: (+972)(4)294369
January 2, 1991

1 Introduction

One of the famous rules for proving partial correctness assertions is the recursion (meta-)rule $REC$ [Hoa71], which is notorious in its apparent circularity. There were several attempts to establish the soundness of a proof system containing it, none of which provided a direct proof. In [DeB80], the Hoare-logic is extended to what is called there "generalized correctness formulae", which include implications between partial correctness assertions. The recursion meta rule is then converted to a proper rule over the generalized formalism, inheriting the natural soundness definition w.r.t. that system. The latter is proved with relation to a denotational semantics, using Scott induction. In [Apt81], another variant of the same approach is employed, using "correctness phrases", through which soundness of a derived rule is proved in terms of a derived Gentzen-like sequent calculus.

In trying to present these indirect proofs to students who are familiar with the usual Hoare rules (related more closely to natural deduction proof systems), difficulties were encountered in explanation and in relating the indirect results to common practice of partial correctness proofs. The basic intuition behind this meta-rule is lost after the above mentioned manipulations.

In this note, we try to provide a direct interpretation of soundness of a system containing a meta-rule, as well as a direct soundness proof, appealing only to concepts that naturally arise in this context.

Our novel idea is to use the provability premiss of the traditional recursion rule to derive proofs (in the underlying proof system without recursion) for all the unfoldings of the recursive procedure. By soundness of the recursion-free system, all the assertions about unfoldings are valid. Hence, any finite computation of the recursive procedure violating partial correctness will induce a
\[
S :: x := e | \text{skip} | S_1 | \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ if } B \text{ do } S_1 \text{ od}
\]

Table 1: Syntax of PLW

1. \((x := e, \sigma)\rightarrow (E, \sigma[e/x])\).
2. \((\text{skip}, \sigma)\rightarrow (E, \sigma)\). 
3. \(\text{if } (S, \sigma)\rightarrow (S', \sigma'), \text{ then for every } T \in \text{PLW } (S; T, \sigma)\rightarrow (S'; T, \sigma')\). 
4. \(\sigma[B] = \text{tt}, \text{ then } (\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}, \sigma)\rightarrow (S_1, \sigma)\). 
5. \(\sigma[B] = \text{ff}, \text{ then } (\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}, \sigma)\rightarrow (S_2, \sigma)\). 
6. \(\sigma[B] = \text{tt}, \text{ then } (\text{while } B \text{ do } S \text{ od, } \sigma)\rightarrow (S, \sigma)\). 
7. \(\sigma[B] = \text{ff}, \text{ then } (\text{while } B \text{ do } S \text{ od, } \sigma)\rightarrow (E, \sigma)\). 

Table 2: The operational semantics of PLW

- Corresponding computation, violating the partial correctness of some unfolding, contradicting the above mentioned validity. 
- To keep the presentation as simple as possible, we confined our treatment to one recursive procedure only, without parameters. The body is assumed to be a while-program (defined below). The extension to more than one procedure, parameters (of simple passing modes) and richer bodies of procedures (e.g., containing nondeterminism via guarded commands) is not hard. 

In Section 2 we discuss some preliminaries. The main proof is in Section 3.

2 Preliminaries

2.1 While programs

As our basic programming language we consider PLW, the language of while-programs. Its syntax is given in Table 1, where \(S\) is a syntactic meta-variable ranging over statements.

The operational semantics is defined by the sos method [HP79, Plo81], providing a transition relation \(\rightarrow\) among configurations.

Definition: A state \(\sigma\) is a mapping from variables to the domain of interpretation \(I\). We use \(\sigma[z]\) to denote the value of variable \(z\) in state \(\sigma\), and \(\sigma[e]\) for the value of expression \(e\) in \(\sigma\). The variant \(\sigma[v/x]\) is a state assigning the value \(v \in I\) to the variable \(x\), and \(\sigma[y]\) to any variable \(y\) different than \(x\). A configuration \(C\) is a pair \(C = (S, \sigma)\), where \(S \in \text{PLW}\) is a syntactic continuation and \(\sigma\) is a state. A configuration is terminal iff \(S = E\).

For the definition of the transition relation, let \(\rightarrow\) be the least relation among configurations, satisfying the clauses in Table 2.

Definition: A computation is a maximal (finite or infinite) sequence of configurations \(C_i, i \geq 0\), such that \(C_i \rightarrow C_{i+1}, i \geq 0\).

Let \(\pi(C_0)\) denote the computation with initial configuration \(C_0\), where \(C_0 = (S_0, \sigma_0)\). 

From the above definitions, we can obtain the following characterization of computations by standard means.
Lemma: (computations)

Atomic statement: If $S_0$ is atomic, then $\pi(C_0)$ has the form $C_0 \rightarrow C_1$, with $C_1$ terminal and $\sigma_1$ conforming to (1) or (2), respectively, in the definition of $\rightarrow$.

Sequential Composition: If $S_0 = S_1; S_2$, then $\pi(C_0)$ has one of the following forms:

1. $C_0 \rightarrow C_1 \rightarrow \ldots$, where $C_i = (T_i; S_2, \sigma_i)$, $i \geq 0$, and $(T_i, \sigma_i)$, $i \geq 0$, is a (nonterminating) computation of $S_1$ on $\sigma_0$.
2. $C_0 \rightarrow C_i = (S_2, \sigma_i) \rightarrow C_{i+1} \rightarrow \ldots$, for some $i \geq 0$, where $C_0 \rightarrow C_i$ induces a finite computation of $S_1$ on $\sigma_0$ and $C_j = (T_j, \sigma_j)$, $j \geq i$, is a (nonterminating) computation of $S_1$.
3. $C_0 \rightarrow (S_2, \sigma') \rightarrow (E, \sigma')$, where $C_0 \rightarrow (E, \sigma')$ induces a finite computation of $S_1$ on $\sigma_0$ and $(S_2, \sigma') \rightarrow (E, \sigma')$ is a computation of $S_2$ on $\sigma'$.

Branching: If $S_0 = \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}$, then $\pi(C_0)$ has one of the following forms:

1. $C_0 \rightarrow C_1 = (S_1, \sigma_1) \rightarrow C_2 \rightarrow \ldots$, where $\sigma_0 \models B$ and $C_i$, $i \geq 1$, is a (nonterminating) computation of $S_1$ on $\sigma_0$.
2. $C_0 \rightarrow C_i = (S_1, \sigma_1) \rightarrow (E, \sigma')$, where $\sigma_0 \models B$ and $C_i \rightarrow (E, \sigma')$ is a (terminating) computation of $S_1$ on $\sigma_1$.
3. Similar 1., with $\neg B$ replacing $B$ and $S_2$ replacing $S_1$.
4. Similar 2., with $\neg B$ replacing $B$ and $S_2$ replacing $S_1$.

Repetition: If $S_0 = \text{while } B \text{ do } S \text{ od}$, then $\pi(C_0)$ has one of the following forms:

1. $C_0 \rightarrow (E, \sigma_0)$, with $\sigma_0 \models \neg B$.
2. $C_0 \rightarrow C_i = (S_0, \sigma_i) \rightarrow \ldots \rightarrow C_k = (S_0, \sigma_k) \rightarrow (C_{k+1}) \rightarrow \ldots$ for some $k \geq 0$, where for $0 \leq j \leq k$ $\sigma_j \models B$, $(S_0, \sigma_j) \rightarrow (S_0, \sigma_{j+1})$ induces a terminating computation of $S$ and $C_{m}, m \geq k$, is a (nonterminating) computation of $S$.
3. $C_0 \rightarrow C_i = (S_0, \sigma_i) \rightarrow \ldots \rightarrow C_k = (S_0, \sigma_k) \rightarrow (S_0, \sigma_{k+1})$ is a computation of $S$.
4. $C_0 \rightarrow C_i = (S_0, \sigma_j) \rightarrow \ldots \rightarrow (E, \sigma_k)$, where for $0 \leq j \leq k$, $\sigma_j \models B$, $\sigma_k \models \neg B$ and $(S_0, \sigma_j) \rightarrow (S_0, \sigma_{j+1})$ is a computation of $S$ for $j < k$.

Definition:

A partial correctness assertion has the form $\{p\} S \{q\}$, where $p$ (the pre-condition) and $q$ (the post-condition) are first-order assertions over $I$. Such an assertion is valid, denoted by $\models I \{p\} S \{q\}$ if and only if for every state $\sigma$, if $\sigma \models I p$ and $(S, \sigma) \rightarrow (E, \sigma')$, then $\sigma' \models I q$.

The rules of the proof system $H$ are presented in Table 2.1.

Let $T_H$ denote all true statements in the interpretation $I$ (used as premises for $CON$).

Theorem: (soundness of $H$)

If $T_H \models I \{p\} S \{q\}$, then $\models I \{p\} S \{q\}$.

There are several proofs of this theorem. The proof w.r.t. the actual operational semantics gives here appears in [Fra91].
\[
\begin{align*}
\{p, x := e \mid p\} & \xRightarrow{Aas} e \\
\{p\} \text{skip} \mid p & \xRightarrow{\text{Skip}} \\
\{p\} S_1 \cdot (r) S_2 \mid q & \xRightarrow{\text{SEQ}} \\
\{p\} S_1; S_2 \mid q & \xRightarrow{\text{COND}} \\
\{p \land B\} S_1 \cdot (q) \cdot \{p \land \neg B\} S_2 \mid q & \xRightarrow{\text{REP}} \\
\{p \land B\} \mid q & \xRightarrow{\text{CONS}}
\end{align*}
\]

Table 3: The proof system \(H\)

2.2 Recursion

We now extend PLW to PROC, a language with parameterless recursive procedures, the bodies of which are expressed in PLW, augmented with procedure invocations. To keep things simple, we restrict the discussion to one procedure only. We add to the syntax the clauses

\[
B::=\text{procedure}\ proc:S
\]

\[
S::=\ldots\mid proc
\]

Here proc ranges over procedure names, and \(S\) is called the body of proc.

As for the operational semantics, we add to the transition rules in Table 2 the extra rule

\[
\{\text{proc},\ σ\} \xrightarrow{} (S, σ)
\]

where \(S\) is the body of proc.

The meta-rule for proving partial correctness of recursive programs, presented in [Hoa71], is shown below.

\[
\text{REC} \quad \frac{T_{\text{H},\{p\} proc \mid q} \vdash \{p\} S \mid q}{\{p\} \text{proc} \mid q}
\]

Note that the assumption \(\{p\} \text{proc} \mid q\) is discharged after the application of the meta-rule.

Denote by \(R\) the proof system \(H \cup \{\text{REC}\}\)

3 A direct soundness proof for the recursion meta-rule

Since a premise of this meta-rule is not a partial correctness assertion, rather a derivability (in \(H\)) assertion, there is no direct way of attributing validity preservation to this rule as is done with ordinary rules in \(H\). However, we still may preserve the definition of soundness of the whole system \(R\) by requiring the validity of every partial correctness assertion derivable in \(R\), ignoring "intermediate" stages in which assumptions are incorporated, as these assumptions are...
discharged at the end of every "proper" proof. We now proceed to establish this
soundness directly, without any extensions of the underlying Hoare-logic.

As a first preparatory step, we define a sequence $S(0)$, $i \geq 0$ of recursion-free
programs that capture what might be called the (consecutive) unfoldings of the
given recursive procedure proc with body $S$. Let $\Omega$ denote the nowhere terminat-
ning program, $\Omega ::= \text{while true do skip od}$. The definition is by induction on

- $S(0) = \Omega$
- $S(i+1) = S[S(i)/\text{proc}]$
  where $S[S'/\text{proc}]$ is obtained from $S\text{PROC}$ by replacing every occurrence of proc with $S'$.

Note that $S(i+1)[\text{proc}/S(0)] = S$ for every $i \geq 0$.

We now relate this sequence with the recursive procedure proc.

Definition:

1. For a computation $\pi = C_i = (S_i, \sigma_i)$, $i \geq 0$, its state projection $\xi(\pi)$
is the sequence of states $\sigma_i$, $i \geq 0$.
2. Two computations (of possibly different programs) correspond if their state
  projections are equal up to finite repetition of identical states.

Lemma: (transition)

Downward transition: For $S_1 \neq \text{proc}; S'$ (for any $S'$), $(S_1, \sigma_1) \rightarrow (S_2, \sigma_2)$ in
proc iff $(S_1[S(n)/\text{proc}], \sigma_1) \rightarrow (S_2[S(n)/\text{proc}], \sigma_2)$, for any $n \geq 0$, in $S(n+1)$.

Upward transition: For $S_1 \neq S(0); S'$ (for any $S'$ and $n \geq 0$), $(S_1, \sigma_1) \rightarrow (S_2, \sigma_2)$
in $S(n+1)$ if $(S_1[\text{proc}/S(n)], \sigma_1) \rightarrow (S_2[\text{proc}/S(n)], \sigma_2)$, in proc.

Proof:

The proof is by induction on the structure of $S_1$, following the cases in the
Computations lemma.

This lemma is easily extended to finite sub-computations by induction on
their lengths, as expressed in the following corollary.

Corollary:

Downward sub-computation: Every finite sub-computation $(S_1, \sigma_1) \rightarrow (S_2, \sigma_2)$
with no invocation of proc, has a corresponding finite sub-computation
$(S_1[S(n)/\text{proc}], \sigma_1) \rightarrow (S_2[S(n)/\text{proc}], \sigma_2)$ of $S(n+1)$.

Upward sub-computation: Every finite sub-computation $(S_1, \sigma_1) \rightarrow (S_2, \sigma_2)$
of $S(n+1)$, with no intermediate configuration of the form $(S(n); S, \sigma)$, has
a corresponding finite sub-computation $(S_1[\text{proc}/S(n)], \sigma_1) \rightarrow (S_2[\text{proc}/S(n)], \sigma_2)$
of proc.

Lemma: (unfolding)

1. If $\pi$ is a finite computation of $(\text{proc}, \sigma_0)$, then there exists some $n \geq 0$ s.t.
   $\pi$ corresponds to a finite computation of $(S(n), \sigma_0)$.
2. Conversely, for any $n > 0$, if $\pi$ is a finite computation of $(S(n), \sigma_0)$, then
   $\pi$ corresponds to a finite computation of $(\text{proc}, \sigma_0)$.

Proof:
1. Let $\pi$ be a finite computation of $(\text{proc}, \sigma_0)$. Hence, it contains only finitely many recursive invocations of $\text{proc}$. We proceed by induction on $r$, the number of recursive invocations of $\text{proc}$ in $\pi$.

**Basis:** $r = 1$.

Let $\pi = (\text{proc}, \sigma_0) \xrightarrow{S_1} (S_1, \sigma_1) \xrightarrow{S_2} \cdots \xrightarrow{S_k} (S_k, \sigma_k)$ be a computation of $\text{proc}$ with only one recursive invocation (in its first transition). By the Downward-transition Corollary,

$$\pi' = (S[S(n) /\text{proc}, \sigma_0] \xrightarrow{S_1[S(n) /\text{proc}, \sigma_1]} \cdots \xrightarrow{S_k[S(n) /\text{proc}, \sigma_k]}$$

is a computation of $S(n+1)$ corresponding to $\pi$.

**Induction step:** Suppose the claim holds for $r \leq m$ and let $\pi$ have $m+1$ recursive calls. By the operational semantics, $\pi$ has the form

$$(\text{proc}, \sigma_0) \xrightarrow{(S, \sigma_1)} (\text{proc}; S', \sigma_1) \xrightarrow{(S', \sigma_1)} (E, \sigma^*)$$

for some $S'$, where $\pi' = (\text{proc}, \sigma'_1) \xrightarrow{S'(n)} (E, \sigma_1)$ is a computation with $m$ recursive invocations only. By the induction hypothesis, there is some $n \geq 1$ s.t. $\pi'$ corresponds to a finite computation $\pi''$ of $(S(n), \sigma_1)$ having the form $(S(n), \sigma_1) \xrightarrow{S'(n)} (E, \sigma_1)$. Since $(S, \sigma_0) \xrightarrow{(\text{proc}; S', \sigma_1)}$ is a sub-computation of $\text{proc}$, we get by the Downward transition Corollary that

$$(S[S(n) /\text{proc}, \sigma_0] \xrightarrow{S_1[S(n) /\text{proc}, \sigma_1]} \cdots \xrightarrow{S_k[S(n) /\text{proc}, \sigma_k]}$$

is a sub-computation of $S(n+1)$. Similarly,

$$(S'[S(n) /\text{proc}, \sigma'_1] \xrightarrow{S'(n)} (E, \sigma^*)$$

is a sub-computation of $S(n+1)$. Consequently,

$$(S(n+1), \sigma_0) \xrightarrow{S(n+1)} (S(n+1), \sigma_0)$$

is the corresponding computation of $(S(n+1), \sigma_0)$.

2. Let $\pi$ be a finite computation of $(S(n), \sigma_0)$, $n \geq 1$. The argument proceeds by induction on $n$.

**Basis:** $n = 1$.

Since $\pi = (S(1), \sigma_0) \xrightarrow{S(1)} (E, \sigma')$ is finite, by the operational semantics it did not reach $\Omega = S(0)$. Hence, by the Upward sub-computation Corollary, there is a finite corresponding computation of $(S, \sigma_0)$ (not invoking $\text{proc}$ recursively), and therefore also a corresponding computation of $(\text{proc}, \sigma_0)$.

**Induction step:** Suppose the claim holds for $S(k)$, $1 \leq k \leq m$, and let $\pi$ be a finite computation of $S(m+1)$. From the definition of $S(m+1)$ it follows that $\pi$ can be presented in the form

$$a_1 \pi_1 \cdots a_l \pi_l a_{l+1}$$

for some $l \geq 0$, where for $1 \leq j \leq l$ the computations $\pi_j$ are finite computations of $S^m$. By the induction hypothesis, each $\pi_j$ has a corresponding computation $\pi'_j$ in $\text{proc}$. Since no $a_l$ contains computations of $S(m)$, by the Upward sub-computation Corollary there are corresponding sub-computations $a'_l$ of $\text{proc}$. The computation

$$\pi' = a'_1 \pi'_1 \cdots a'_l \pi'_l a'_{l+1}$$

is a computation of $\text{proc}$ corresponding to $\pi$.  

6
Proposition: If
\[ T_{\mathcal{H}} \triangleright_{\mathcal{H}} (p) \text{ proc } (q) \vdash (p) S (q), \]
then for every \( n \geq 0, \)
\[ T_{\mathcal{H}} \triangleright_{\mathcal{H}} (p) S^{(n)} (q) \vdash (p) S^{(n+1)} (q) \]

Proof:
Consider any application of a rule in \( \mathcal{H} \) using \( (p) \text{ proc } (q) \) as one of its premises. In its conclusion, there is some occurrence of proc in the program part. By the definition of \( S^{(n+1)} \), the corresponding program section is obtained by replacing that occurrence with \( S^{(n)} \). Thus, the same rule can be applied in the proof of \( (p) S^{(n+1)} (q) \) by using the assumption \( (p) S^{(n)} (q) \).

For example, consider a proof fragment of \( T_{\mathcal{H}} \triangleright_{\mathcal{H}} (p) \text{ proc } (q) \)
where the recursive invocation of proc is sequentially composed, say with an assignment statement \( z := e \).

\begin{align*}
(i) & \quad (p) \text{ proc } (q) \quad \text{Assumption} \\
(j) & \quad ((p')^* x := e (p') \quad \text{Ass} \\
(k) & \quad r \Rightarrow p \quad \text{Logic} \\
(h) & \quad ((p')^* x := e (p) \quad j, k, \text{CONS} \\
(g) & \quad (p'^* x := e) \quad \text{proc}(q) \quad i, k, \text{SEQ}
\end{align*}

If we replace line (i) by
\[(i) \quad (p) S^{(n)} (q) \]
which is assumed, we can derive
\[(g) \quad ((p')^* x := e; S^{(n)} (q) \quad i, k, \text{SEQ}
\]
as before.

By an inductive argument, using the fact that \( \vdash (p) \Omega (q) \) as a basis, and the proposition as the induction step, we obtain:
Conclusion:
If \( T_{\mathcal{H}} \triangleright_{\mathcal{H}} (p) \text{ proc } (q) \vdash (p) S (q) \)
then, for every \( n \geq 0, T_{\mathcal{H}} \triangleright_{\mathcal{H}} (p) S^{(n)} (q) \).

Finally, by the soundness of \( \mathcal{H} \), we have:
Conclusion:
If \( T_{\mathcal{H}} \triangleright_{\mathcal{H}} (p) \text{ proc } (q) \vdash (p) S (q) \).
then, \( \vdash (p) S^{(n)} (q) \).
These conclusions lead to the main result of the paper.

Theorem: (soundness of R)
For \( P \in \text{PROC}, \) if \( T_{\mathcal{R}} \triangleright_{\mathcal{R}} (p) P (q) \) then \( \vdash (p) P (q) \).

Proof:
The only interesting case is when \( (p) \text{ proc } (q) \) is deduced by means of R.
Assume the premise of the rule holds, and suppose that it is not the case that \( \vdash (p) \text{ proc } (q) \). Clearly, every computation of proc which generates infinitely many recursive invocations of the procedure proc is nonterminating, and hence cannot violate the partial correctness by definition. Thus, we may assume that there exists a terminating computation of proc, starting in a configuration with an initial state \( s_0 \) such that \( s_0 \vdash (p \), and ends in a terminal configuration with
a state $\sigma^*$, such that $\sigma^* I_{\underline{q}} q$ does not hold. By the unfolding lemma, this computation is also a finite computation of some $\sigma^{(\ell)}$, violating the partial correctness of the latter w.r.t. the same specification, as obtained from the second conclusion above (under the assumption that $REC$ was successfully applied).

ACKNOWLEDGMENTS

The part of the first author was partially supported by a grant for basic research by the Israeli Academy of Sciences and Humanities and by the Fund for the Promotion of Research in the Technion.

References


In E. Engeler, editor, Symposium on semantics of algorithmic languages,

a simple programming language. LNCS 74, Springer-Verlag, New York,
1979.

Technical Report DAIMI FN-10, Computer science dept., Aarhus
univ., 1981.