CHAPTER 11

Total Correctness

"For He will finish the work and cut it short in righteousness,
Because the LORD will make a short work upon the earth."
— Romans 9:28

"For the Lord GOD of hosts
Will make a determined end
In the midst of all the land."
— Isaiah 10:23

The proof of the termination of programs, and hence their total correctness, is presented in this chapter. We start with the assumptions of partial correctness, precondition maintenance, conditional termination, and most importantly, recursiveness, and prove the termination of every call of every procedure declared in the mutually recursive procedure environment. This leads to an environment which has been verified to be well-formed for total correctness, and thus to be fully well-formed. The total correctness of the environment becomes the last essential element in the proof of the ultimate theorem of this work, Theorem 7.12, as presented in Chapter 7, that the verification condition generator has been

verified for total correctness.

Total Correctness has two aspects, partial correctness and termination. In the past these have sometimes been proven apart from each other, and sometimes together, often using the same overall proof structure. But there has begun to appear evidence that there is a more substantial difference between partial correctness and termination than had originally been thought, when recursive procedures are present. In 1990, America and de Boer reported [AdB90]

> ...we may conclude that reasoning about total correctness differs from partial correctness in a substantial way which has not been recognized til now.

In the course of this work, this difference has been exposed and explored. It became evident during the construction of the verification of the VCG that partial correctness was a necessary precursor to even beginning the attack on total correctness. Many of the rules presented in Chapter 6 in the entrance logic and in the termination logic contained partial correctness specifications as necessary antecedents. Moreover, to prove the environment was well-formed for recursion, it was necessary first to have the entire environment established to be well-formed for partial correctness and for calls progress. In a similar way, we will add the assumption that the environment is well-formed for conditional termination, and prove from these that the environment is well-formed for total correctness.

We have already seen a substantial argument was in order to prove the full recursiveness property for procedures, how it was necessary to introduce the entrance logic in order to verify the progress claimed in the *calls* progress expressions

in the headers of procedures, and how it was necessary to introduce the analysis of the call graph structure to verify that the progress claimed in the recursion expressions were supported by the progress of the *calls* progress expressions. We also saw it was necessary to introduce the termination logic in order to verify the conditional termination of commands. Now all of these elements come together as necessary precursors to the proof of termination of every procedure. This extended proof, with these layers and stages of development, demonstrates the depth of reasoning that is necessary to prove termination. The good part of this is that once done, it need not be repeated when the VCG is applied. The verification of the VCG allows it to be used without repeating the intricate arguments expressed and proven at the meta level here.

We will begin by summarizing the substance of the argument up to this point.

11.1 Reprise

11.1.1 Entrance Logic

In Section 6.3, we presented an Entrance Logic, including correctness specifications of the forms

$\{a_1\}\ c \rightarrow p\ \{a_2\}\ /\rho$	entrance specification
$\{a\}\ c \rightarrow \mathbf{pre}\ /\rho$	precondition entrance specification
$\{a\}\ c \rightarrow calls\ /\rho$	calls entrance specification
$\{a_1\}\ p_1 - ps \rightarrow p_2\ \{a_2\}\ /\rho$	path entrance specification
$\{a_1\}\ p \hookleftarrow \{a_2\}\ /\rho$	recursive entrance specification

We then presented the rules of the Entrance Logic which supported proofs of

these correctness specifications for specific program fragments. Later we saw how these rules supported the verification of parts of the VCG, that the truth of the verification conditions produced by the syntax-directed analysis of a procedure's body sufficed to guarantee the partial correctness of the body with respect to the given precondition and postcondition, to guarantee the progress claimed by the *calls* specifications, and to guarantee the achievement of the preconditions of every called procedure at their entrance.

11.1.2 Termination Logic

In Section 6.4, we presented a Termination Logic, including correctness specifications of the forms

$[a]\ c \downarrow /\rho$	command conditional termination specification
$p \downarrow /\rho$	procedure conditional termination specification
$[a]\ c \Downarrow /\rho$	termination specification

We then presented the rules of the Termination Logic which supported proofs of these correctness specifications for specific program fragments. Later we saw how these rules supported the verification of parts of the VCG, that the truth of the verification conditions produced by the syntax-directed analysis of a procedure's body sufficed to guarantee the conditional termination of that body, given the termination of every procedure called immediately from that body.

11.1.3 Recursiveness

Given the properties proven about the environment of all defined procedures, that it was well-formed for partial correctness, precondition maintenance, calls

progress, and conditional termination, we showed in Section 7.1.3 a series of functions defined as part of the VCG that analyzed the procedure call graph and produced a list of verification conditions, whose proof, along with the progress claimed by the *calls* specifications previously shown, was sufficient to prove the full recursiveness property, that every recursive call evidenced the progress claimed in the recursion expression for that procedure.

That progress expressed in the form $v < x$, that the value of the expression v strictly decreased from the initial call to the recursive call. This was an example of an expression whose value was a member of a well-founded set, in this case the nonnegative integers. Well-founded sets have the property that there are no infinitely decreasing sequences of values from the set. This lays the foundation for the argument for termination, that if there were a procedure call that exhibited infinite recursive descent, then taking the sequence of values of v at each recursive entrance of the procedure would exhibit such an infinitely decreasing sequence. Since that is excluded by the definition of well-founded sets, there cannot be such a nonterminating procedure call.

11.2 Termination

We will now present the main points of our proof of the termination of mutually recursive procedures. We begin by defining two more semantic relations.

These semantic relations, $terminates$ and $Depth\_calls$, are defined in Tables 11.1 and 11.2. These are related to the semantic relations defined in Chapter 5. First, $terminates$ expresses the condition that a particular procedure's body terminates when started in a given state. Then $Depth\_calls$ connects a procedure

name and a state to another procedure name and a state, where there is an execution sequence between the first state at the entrance of the first procedure through nested calls to the second state at the entrance of the second procedure. Of particular interest is that *Depth_calls* specifies the length of the chain of calls as a particular integer. Thus *Depth_calls* provides a way to describe calls which are nested a particular number of calls deep from the original point where the execution began.

$$
\begin{array}{l}
terminates\ p\ s\ \rho\ = \\
\qquad \textbf{let}\ \langle vars, vals, glbs, pre, post, calls, rec, c\rangle = \rho\ p\ \textbf{in} \\
\qquad (\exists s'.\ C\ c\ \rho\ s\ s')
\end{array}
$$

Table 11.1: Termination Semantic Relation *terminates*.

$$
\begin{array}{l}
Depth\_calls\ 0\ p_1\ s_1\ p_2\ s_2\ \rho\ = \\
\qquad p_1 = p_2\ \wedge\ s_1 = s_2 \\
\\
Depth\_calls\ (n+1)\ p_1\ s_1\ p_2\ s_2\ \rho\ = \\
\qquad \exists p_3\ s_3.\ M\_calls\ p_1\ s_1\ [\,]\ p_3\ s_3\ \rho\ \wedge \\
\qquad\qquad Depth\_calls\ n\ p_3\ s_3\ p_2\ s_2\ \rho
\end{array}
$$

Table 11.2: Termination Semantic Relation *Depth_calls*.

11.2.1 Sketch of Proof

We will first give an sketch of our proof of termination, and then develop that sketch in detail.

SKETCH:

Every command terminates if all of its immediate calls terminate. Hence, it follows that every procedure body terminates if for any n, all of the body's calls at depth n or less terminate. Thus, to show a procedure body terminates, it suffices to show there is an n such that all of the body's calls of depth n or less terminate.

Assume the opposite, that for some procedure body and initial state, that for all n, there is some call at depth n or less which does not terminate. Then there is some call at depth n which does not terminate, for all n. This implies there exists an infinite sequence of nested procedure calls issuing from the original procedure body and state which do not terminate. Consider this sequence of procedures which are called and the states at their entrances. There must be some procedure which occurs an infinite number of times in this sequence, or else the sequence could not itself be infinite, since there is only a finite number of declared procedures. Let p be such a procedure that occurs an infinite number of times, and let v be its recursion expression. Form the infinite sequence of the values of v in the states at every occurrence of p in the first sequence. By the recursiveness property, we have that every pair of values in this sequence is strictly decreasing, and hence this sequence is strictly decreasing. This is then an infinite sequence of decreasing values. But since the set of nonnegative integers is a well-founded set, no such infinite decreasing sequence can exist. Hence our original assumption was wrong, and we may conclude the opposite, that for some n, all of the original procedure's body's calls at depth n or less terminate. As we have shown above, this then implies that the procedure body terminates,

unconditionally.

The termination of procedure bodies, combined with the termination of commands based on their immediate calls terminating, gives us that all commands terminate unconditionally. Combining this with the partial correctness of commands gives us the total correctness of commands. The total correctness of commands implies the total correctness of procedure bodies, and hence the entire environment is proved to be fully well-formed.

End of **SKETCH.**

We will now elaborate the sketch.

11.2.2 Termination of Deep Calls

First, we have already shown that every command terminates if all of its immediate calls terminate. That is, consider a command c begun execution in a state s_1. Let s_2 be any possible state which is reachable from s_1 by being the state at the entrance of a procedure called immediately from c. If for all such s_2, the body of that procedure when begun in s_2 terminates, then c must terminate. This last statement is guaranteed by the definition of WF_{env_term}, which is verified to hold based on the syntax-directed part of the VCG and the verification conditions it produces, by the theorem `vcgd_TERM`, given in Table 7.3. It is the primary starting point for the rest of this argument.

Since every command terminates if all of its immediate calls terminate, this also applies to the commands which are the bodies of procedures. Therefore every procedure body terminates if all of its immediate calls terminate. But then consider those immediate calls. Each one of those causes the execution

of a procedure body, whose termination is implied by the termination of *its* immediate calls. We may then restate this, that the original procedure body would be guaranteed of terminating if all of the procedure calls at the second level down terminate. More generally, if the original body terminates if all calls at the nth level terminate, then since each one of those calls at the nth level terminates if all their immediate calls terminate, we may say that the original body terminates if all calls at the $(n+1)$th level terminate. Then by induction on n, we say that for any n, if the calls at depth n terminate, then the original body terminates.

In Table 11.3, we have proven that a call at one depth implies that there exist calls at all lesser (more shallow) depths.

$$
\begin{array}{l}
\vdash \forall n\ m\ p_1\ s_1\ p_2\ s_2\ \rho. \\
\qquad WF_{env\_term}\ \rho \wedge WF_{env\_pre}\ \rho \wedge \\
\qquad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ p_1))\ s_1 \wedge \\
\qquad Depth\_calls\ n\ p_1\ s_1\ p_2\ s_2\ \rho \wedge \\
\qquad m \leq n\ \Rightarrow \\
\qquad (\exists p_3\ s_3. Depth\_calls\ m\ p_1\ s_1\ p_3\ s_3\ \rho)
\end{array}
$$

Table 11.3: Theorem of existence of shallower calls.

We have as a theorem in Table 11.4 that if all the calls at one depth or less terminate, then the original procedure call terminates. Since the termination of all the calls at one depth implies the termination of all the calls at one less depth, then by induction we can prove the termination of all calls at shallower depth.

Contrariwise, in Table 11.5 we have proven that if a call at one depth from the original call does not terminate, then for all greater depths, there is a call at that depth from the original call. This is valuable, but it does not yet give us the

$$
\begin{array}{l}
\vdash \forall \rho.\ \ WF_{env\_term}\ \rho \wedge WF_{env\_pre}\ \rho \Rightarrow \\
\qquad (\forall n\ p_1\ s_1.\ (\forall m\ p_2\ s_2.\ m \leq n \wedge Depth\_calls\ m\ p_1\ s_1\ p_2\ s_2\ \rho \Rightarrow \\
\qquad\qquad\qquad terminates\ p_2\ s_2\ \rho) \wedge \\
\qquad\qquad (\textbf{let}\ \langle vars, vals, glbs, pre, post, calls, rec, c\rangle = \rho\ p_1\ \textbf{in} \\
\qquad\qquad\qquad A\ pre\ s_1) \Rightarrow \\
\qquad\qquad terminates\ p_1\ s_1\ \rho
\end{array}
$$

Table 11.4: Theorem of termination of shallower calls.

existence of an infinite chain of calls, because it does not include the condition that every procedure and state in the chain actually arose from a call from the previous procedure and state.

$$
\begin{array}{l}
\vdash \forall m\ n\ p_1\ s_1\ p_2\ s_2\ \rho. \\
\qquad WF_{env\_term}\ \rho \wedge WF_{env\_pre}\ \rho \wedge \\
\qquad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ p_1))\ s_1 \wedge \\
\qquad Depth\_calls\ n\ p_1\ s_1\ p_2\ s_2\ \rho \wedge \\
\qquad \sim(terminates\ p_2\ s_2\ \rho)\ \Rightarrow \\
\qquad (\exists p_3\ s_3. Depth\_calls\ (n+m)\ p_1\ s_1\ p_3\ s_3\ \rho\ \wedge\ \sim(terminates\ p_3\ s_3\ \rho))
\end{array}
$$

Table 11.5: Theorem of existence of all deeper calls.

11.2.3 Existence of an Infinite Sequence

In the termination proof sketch, at one point we assume that there does not exists any n such that all calls of depth n or less terminate. Then for all n there must be some call at depth n or less which does not terminate. This then should imply the existence of an infinite sequence of deeper and deeper calls.

We will prove the existence of such an infinite sequence by actually constructing and exhibiting one. First, we define the function *mk_sequence* in Table 11.6 as a generator function to take a pair $\langle p, s \rangle$ of a procedure name and a state, and return the next $\langle p', s' \rangle$ of the infinite sequence.

$$
\begin{array}{l}
mk\_sequence\ 0\ \rho\ p\ s\ =\ p, s \\
\\
mk\_sequence\ (i+1)\ \rho\ p\ s\ = \\
\qquad \textbf{let}\ (p', s')\ =\ @\ (p', s').\ \ M\_calls\ p\ s\ [\,]\ p'\ s'\ \rho\ \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad \sim(terminates\ p'\ s'\ \rho) \\
\qquad \textbf{in}\ mk\_sequence\ i\ \rho\ p'\ s'
\end{array}
$$

Table 11.6: Sequence Generator Function *mk_sequence*.

Here @ is the Hilbert choice operator, which returns some element of its range type which satisfies the given condition, if any elements do satisfy it. If none do, then @ still chooses some arbitrary element. This is a total function, so it always returns the same choice, but all that is known about the element chosen is the property specified, and that only if there exists such an element.

Given this definition, we can prove that it is well-defined, in the sense that every pair of the sequence satisfies the definition property, as in Table 11.7.

$$
\begin{array}{l}
\vdash \forall i\ p_1\ s_1\ \rho. \\
\qquad WF_{env\_term}\ \rho \wedge WF_{env\_pre}\ \rho\ \wedge \\
\qquad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ p_1))\ s_1\ \wedge \\
\qquad \sim(terminates\ p_1\ s_1\ \rho)\ \Rightarrow \\
\qquad\ \textbf{let}\ (p_2, s_2) = mk\_sequence\ i\ \rho\ p_1\ s_1\ \textbf{in} \\
\qquad\ (Depth\_calls\ i\ p_1\ s_1\ p_2\ s_2\ \rho\ \wedge\ \sim(terminates\ p_2\ s_2\ \rho))
\end{array}
$$

Table 11.7: Definitional property satisfied by *mk_sequence*.

The most important property we prove about *mk_sequence* is that the sequence it generates is chained together by each consecutive pair being related by one level of procedure call, as expressed in Table 11.8.

$$
\begin{array}{l}
\vdash \forall i\ p_1\ s_1\ \rho. \\
\quad WF_{env\_term}\ \rho \wedge WF_{env\_pre}\ \rho \wedge \\
\quad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ p_1))\ s_1 \wedge \\
\quad \sim(terminates\ p_1\ s_1\ \rho)\ \Rightarrow \\
\quad\ \textbf{let}\ (p_2, s_2) = mk\_sequence\ i\ \rho\ p_1\ s_1\ \textbf{in} \\
\quad\ \textbf{let}\ (p_3, s_3) = mk\_sequence\ (i+1)\ \rho\ p_1\ s_1\ \textbf{in} \\
\quad\ M\_calls\ p_2\ s_2\ [\,]\ p_3\ s_3\ \rho
\end{array}
$$

Table 11.8: Chain of calls induced by *mk_sequence*.

Given this generator function *mk_sequence*, it is possible to prove that the sequence of procedure names and states it generates satisfies the properties in Table 11.9 to be called an infinite recursive descent sequence.

$$
\begin{array}{l}
sequence\ ps\ sts\ ns\ \rho\ = \\
\quad (\forall i.\ M\_calls\ (ps\ i)\ (sts\ i)\ [\,]\ (ps\ (i+1))\ (sts\ (i+1))\ \rho) \wedge \\
\quad (\forall i.\ ns\ i\ =\ (\textbf{let}\ \langle vars, vals, glbs, pre, post, calls, rec, c\rangle = \rho\ (ps\ i)\ \textbf{in} \\
\qquad\qquad\qquad\qquad induct\_start\_num\ (sts\ i)\ rec))
\end{array}
$$

Table 11.9: Infinite Recursive Descent Sequence Predicate *sequence*.

In this definition, *ps* is an infinite sequence of procedure names, represented as a function from `num` to `string`. The number used as the index is the depth number from *Depth_calls*. *ps* contains the infinite sequence of names of procedures called in the hypothesized infinite recursive descent; it is the path downward.

Likewise, sts is the corresponding infinite sequence of states, each one the state reached in the corresponding procedure in ps in the process of the infinite recursive descent.

Finally, ns is the corresponding infinite sequence of the values of the recursion expressions of each procedure in ps, evaluated in the corresponding state given in sts. Several procedures may be represented in this list; we shall see that for the subsequences of this sequence for any particular procedure, each such subsequence will be strictly decreasing. The values in ns are generated by the function $induct\_start\_num$, defined in Table 11.10.

$induct\_start\_num\ s\ \mathbf{false}\ =\ 0$
$induct\_start\_num\ s\ (v < x)\ =\ V\ v\ s$

Table 11.10: Recursion Expression Value Function $induct\_start\_num$.

This gives the definition of an infinite recursive descent sequence. Such a sequence is implied by the assumption stated earlier, that there does not exist any n such that all calls of depth n or less terminate. We can now prove this as the theorem listed in Table 11.11, using $mk\_sequence$ to create an explicit witness.

$$
\begin{array}{l}
\vdash \forall p_1\ s_1\ \rho. \\
\quad WF_{env\_term}\ \rho \wedge WF_{env\_pre}\ \rho\ \wedge \\
\quad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ p_1))\ s_1\ \wedge \\
\quad \sim(\exists n.\ \forall m\ p_2\ s_2.\ m \leq n\ \wedge\ Depth\_calls\ m\ p_1\ s_1\ p_2\ s_2\ \rho \Rightarrow \\
\qquad\qquad terminates\ p_2\ s_2\ \rho)\ \Rightarrow \\
\quad (\exists ps\ sts\ ns.\ sequence\ ps\ sts\ ns\ \rho\ \wedge\ (ps\ 0 = p_1)\ \wedge\ (sts\ 0 = s_1))
\end{array}
$$

Table 11.11: Existence of Infinite Recursive Descent Sequence.

11.2.4 Consequences of an Infinite Sequence

One consequence of the definition of an infinite recursive descent sequence is that any two points in the sequence are related by M_calls; this is displayed in Table 11.12.

$$
\begin{array}{l}
\vdash \forall \rho\ ps\ sts\ ns\ i\ j. \\
\quad sequence\ ps\ sts\ ns\ \rho\ \wedge\ i < j\ \Rightarrow \\
\quad (\exists ps'.\ M\_calls\ (ps\ i)\ (sts\ i)\ ps'\ (ps\ j)\ (sts\ j)\ \rho)
\end{array}
$$

Table 11.12: Sequence calls related by M_calls.

Preconditions are maintained across points in the sequence, as in Table 11.13.

$$
\begin{array}{l}
\vdash \forall i\ \rho\ ps\ sts\ ns. \\
\quad WF_{env\_pre}\ \rho\ \wedge \\
\quad sequence\ ps\ sts\ ns\ \rho\ \wedge \\
\quad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ (ps\ 0)))\ (sts\ 0)\ \Rightarrow \\
\quad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ (ps\ i)))\ (sts\ i)
\end{array}
$$

Table 11.13: Sequence Precondition Maintenance.

11.2.5 Strictly Decreasing Sequences

Perhaps the most important consequence of an infinite recursive descent sequence results from combining it with the knowledge contained in the recursiveness property, WF_{env_rec}. WF_{env_rec} says that every recursive call of a procedure exhibits the strict decrease of the value of its recursion expression. For sequences, this gives us the ability to prove the theorem in Table 11.14. This says that for any two points in the infinite sequence which refer to the *same* procedure, the value of the recursion expression as stored in ns strictly decreases.

$$
\begin{array}{l}
\vdash \forall \rho\ ps\ sts\ ns\ p\ i\ j\ vars\ vals\ glbs\ pre\ post\ calls\ rec\ c. \\
\qquad WF_{env\_syntax}\ \rho\ \wedge \\
\qquad WF_{env\_pre}\ \rho\ \wedge \\
\qquad WF_{env\_rec}\ \rho\ \wedge \\
\qquad sequence\ ps\ sts\ ns\ \rho\ \wedge \\
\qquad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ (ps\ 0)))\ (sts\ 0)\ \wedge \\
\qquad (\rho\ p = \langle vars, vals, glbs, pre, post, calls, rec, c \rangle)\ \wedge \\
\qquad (ps\ i = p)\ \wedge \\
\qquad (ps\ j = p)\ \wedge \\
\qquad i < j\ \Rightarrow \\
\qquad (ns\ j < ns\ i)
\end{array}
$$

Table 11.14: Sequence Decreasing Values.

To make use of this strictly decreasing property, we choose a minor variation on the proof sketch described earlier. Instead of claiming that there must be some procedure which has an infinite number of occurrences in the sequence, we take the approach of proving that every procedure has only a finite number of occurrences in the sequence. We first prove that given any occurrence of a procedure p in the sequence, there is a maximum limit on the index of the

elements beyond which none of the elements refer to that same procedure p, as shown in Table 11.15.

$$
\begin{array}{l}
\vdash \forall n\ i\ p\ \rho\ ps\ sts\ ns\ vars\ vals\ glbs\ pre\ post\ calls\ rec\ c. \\
\qquad WF_{env\_syntax}\ \rho\ \wedge \\
\qquad WF_{env\_pre}\ \rho\ \wedge \\
\qquad WF_{env\_rec}\ \rho\ \wedge \\
\qquad sequence\ ps\ sts\ ns\ \rho\ \wedge \\
\qquad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ (ps\ 0)))\ (sts\ 0)\ \wedge \\
\qquad (\rho\ p = \langle vars, vals, glbs, pre, post, calls, rec, c \rangle)\ \wedge \\
\qquad (ps\ i = p)\ \wedge \\
\qquad (ns\ i = n)\ \Rightarrow \\
\qquad (\exists m.\ \forall j.\ m < j \Rightarrow ps\ j \neq p)
\end{array}
$$

Table 11.15: Sequence Occurrence Implies Limit on Occurrences.

This is proven by well-founded induction on n, the value of the recursion expression at the ith procedure in the sequence, making use of the fact that the values of the recursion expression are members of a well-founded set.

From this we are able to prove that for every procedure, there is a maximum limit on the index of the elements which refer to it, as shown in Table 11.16.

$$
\begin{array}{l}
\vdash \forall p\ \rho\ ps\ sts\ ns. \\
\qquad WF_{env\_syntax}\ \rho\ \wedge \\
\qquad WF_{env\_pre}\ \rho\ \wedge \\
\qquad WF_{env\_rec}\ \rho\ \wedge \\
\qquad sequence\ ps\ sts\ ns\ \rho\ \wedge \\
\qquad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ (ps\ 0)))\ (sts\ 0)\ \Rightarrow \\
\qquad (\exists m.\ \forall j.\ m < j \Rightarrow ps\ j \neq p)
\end{array}
$$

Table 11.16: Each Procedure Has Limit on Occurrences.

Next we need to establish that every procedure in the sequence is a member

of the finite list of defined procedures, all_ps, as described in Table 11.17.

$$
\begin{array}{l}
\vdash \forall all\_ps\ \rho\ ps\ sts\ ns. \\
\qquad WF_{env\_pre}\ \rho\ \wedge \\
\qquad (\forall p.\ p \notin SL\ all\_ps \Rightarrow \rho\ p = \rho_0\ p)\ \wedge \\
\qquad sequence\ ps\ sts\ ns\ \rho\ \wedge \\
\qquad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ (ps\ 0)))\ (sts\ 0)\ \Rightarrow \\
\qquad (\forall i.\ ps\ i \in SL\ all\_ps)
\end{array}
$$

Table 11.17: Each Procedure in Sequence is in all_ps.

We can now prove that since for each procedure there is a maximum limit on its occurrences in ps, and since there is only a finite number of procedures, there must be a maxiumum limit on the sequence as a whole. This means there exists a single limit m which bounds the indices of the occurrences of *all* the procedures listed in all_ps, as in Table 11.18.

$$
\begin{array}{l}
\vdash \forall all\_ps\ \rho\ ps\ sts\ ns. \\
\qquad WF_{env\_syntax}\ \rho\ \wedge \\
\qquad WF_{env\_pre}\ \rho\ \wedge \\
\qquad WF_{env\_rec}\ \rho\ \wedge \\
\qquad sequence\ ps\ sts\ ns\ \rho\ \wedge \\
\qquad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ (ps\ 0)))\ (sts\ 0)\ \Rightarrow \\
\qquad (\exists m.\ \forall p.\ p \in SL\ all\_ps \Rightarrow (\forall j.\ m < j \Rightarrow ps\ j \neq p))
\end{array}
$$

Table 11.18: Limit on All Occurrences in all_ps.

This then contradicts the assumption of the infinite sequence, since there are many elements beyond the maximum limit, and they must belong to *some* defined procedure. This contradiction is expressed in Table 11.19.

Given this contradiction, implied by the assumption that there did not exist

$$
\begin{array}{l}
\vdash \forall \rho\ all\_ps\ ps\ sts\ ns. \\
\qquad WF_{env\_syntax}\ \rho\ \wedge \\
\qquad WF_{env\_pre}\ \rho\ \wedge \\
\qquad WF_{env\_rec}\ \rho\ \wedge \\
\qquad (\forall p.\ p \notin SL\ all\_ps \Rightarrow \rho\ p = \rho_0\ p)\ \wedge \\
\qquad A\ ((FST \circ SND \circ SND \circ SND)\ (\rho\ (ps\ 0)))\ (sts\ 0)\ \Rightarrow \\
\qquad \sim (sequence\ ps\ sts\ ns\ \rho)
\end{array}
$$

Table 11.19: Sequence Contradiction.

any n such that all calls of depth n or less terminated, we can conclude that such an n must exist, and hence by the theorem in Table 11.4, we can prove that every procedure terminates, as shown in Table 11.20.

$$
\begin{array}{l}
\vdash \forall \rho\ all\_ps\ p\ s\ vars\ vals\ glbs\ pre\ post\ calls\ rec\ c. \\
\qquad WF_{env\_syntax}\ \rho\ \wedge \\
\qquad WF_{env\_pre}\ \rho\ \wedge \\
\qquad WF_{env\_rec}\ \rho\ \wedge \\
\qquad WF_{env\_term}\ \rho\ \wedge \\
\qquad (\forall p.\ p \notin SL\ all\_ps \Rightarrow \rho\ p = \rho_0\ p)\ \wedge \\
\qquad (\rho\ p = \langle vars, vals, glbs, pre, post, calls, rec, c \rangle)\ \wedge \\
\qquad A\ pre\ s\ \Rightarrow \\
\qquad terminates\ p\ s\ \rho
\end{array}
$$

Table 11.20: Procedure Termination.

Finally, given the termination of each procedure when called, we can prove the total correctness of the entire environment of procedures, as in Table 11.21.

$$
\begin{array}{l}
\forall d\ \rho. \\
\quad \rho = mkenv\ d\ \rho_0\ \wedge \\
\quad WF_{env\_syntax}\ \rho\ \wedge \\
\quad WF_{env\_pre}\ \rho\ \wedge \\
\quad WF_{env\_rec}\ \rho\ \wedge \\
\quad WF_{env\_term}\ \rho\ \Rightarrow \\
\quad WF_{env\_total}\ \rho
\end{array}
$$

Table 11.21: Total Correctness of Procedure Environment.

This completes our proof of termination for the Sunrise language.

Part IV

Conclusions