A semantics for while with break, continue and goto

Patricia Peratto
psperatto@adinet.com.uy
Uruguay
August 2006

Abstract

This work presents a formal description of a subset of a C-like language in the form of an operational semantics. We give semantics to the following statements (presented in alphabetical order): assignment, break, composition, continue, goto, if, skip and while. The semantics is given by an abstract machine composed by a stack, two counters and three functions. We prove some expected properties of the semantics.

1 Introduction

There is few work concerning the operational semantics of statements break, continue and goto for C-like languages. Previous work does not treat these statements or take a different point of view. In particular the sentence goto was not treated at all.

The two counters of our abstract machine index while-loops. Using them together with the second function, we can know when an instruction break, continue or goto is inside an active while-loop (we say a while-loop is active when its boolean condition evaluates to true).

The stack works as a continuation, we put in it the sentences that have to be executed after the actual one.

The first of the functions maps labels to environments and using it we give meaning to goto statements.

The third function represents an state in which we store the values of program variables.

The outline of our paper is as follows: section 2 presents the abstract syntax of the language (this includes arithmetic and boolean expressions and
Section 3 presents the semantics of expressions and section 4 the semantics of statements. We prove the semantics follows the intended meaning of the statements \texttt{continue}, \texttt{break} and \texttt{goto}. In section 5 we prove the semantics is deterministic. In section 6 we present related work and in section 7 conclusions.

2 Abstract syntax of the language

We use a syntactic notation based on BNF. Parenthesis can be used (not indicated in our BNF) to solve ambiguities and uniquely determine the corresponding parse tree.

We have the following Syntactic Categories and meta-variables ranging over them:

- $n$ will range over numerals, \texttt{Num}
- $a$ will range over arithmetic expressions, \texttt{AExp}
- $x$ will range over arithmetic variables, \texttt{AVar}
- $b$ will range over boolean expressions, \texttt{BExp}
- $x$ will range over boolean variables, \texttt{BVar}
- $S$ will range over statements, \texttt{Stm}

The meta-variables can be primed or subscripted for example $a, a', a_1, a_2$ all stand for arithmetic expressions. $x$ refers both to arithmetic and boolean variables. What we mean can be inferred from the context.

\textbf{Definition 2.1.} Abstract syntax for Arithmetic Expressions

\[ a ::= n \mid x \mid a_1 + a_2 \mid a_1 \times a_2 \mid a_1 - a_2 \]

\textbf{Definition 2.2.} Abstract syntax for Boolean Expressions

\[ b ::= \text{true} \mid \text{false} \mid x \mid a_1 = a_2 \mid a_1 < a_2 \mid \neg b \mid b_1 \lor b_2 \mid b_1 \land b_2 \]

\texttt{true} and \texttt{false} stand for constant truth values.

\textbf{Definition 2.3.} Abstract syntax for Statements
\[ S ::= x := a \mid x := b \mid \text{skip} \mid S_1 ; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S \mid \text{continue} \mid \text{break} \mid \text{label : } S \mid \text{goto } \text{label} \]

label stands for an identifier.

3 Semantics of expressions

We define the following semantic functions:

\[ A : \text{Aexp} \rightarrow (\text{State} \rightarrow \mathbb{Z}) \text{ for arithmetic expressions} \]

\[ B : \text{Bexp} \rightarrow (\text{State} \rightarrow \mathbb{T}) \text{ for boolean expressions} \]

where

\[ \mathbb{Z} \text{ is the semantic domain of integers ranged over by metavariables } n, z, \]

\[ \mathbb{T} \text{ is the semantic domain of the truth values ranged over by metavariable } t \]

and

\[ \text{State} = (\text{AVar} + \text{BVar}) \rightarrow (\mathbb{Z} + \mathbb{T}) \]

is a semantic function (from syntactic domains to semantic domains) ranged over by s.

+ denotes disjoint union, with constructors \text{inl} and \text{inr} both over syntactic domains and over semantic domains. \( s \in \text{State} \) satisfies \( s(\text{inl } x) \in \mathbb{Z} \) and \( s(\text{inr } x) \in \mathbb{T} \).

Definition 3.1. Semantics of Arithmetic Expressions

The transition relation is specified by the rules of the transition system presented in Table 1.

\(+, \ast, -\) at the left denote syntax and at the right the corresponding operations over the semantical domains.

We define the semantic function

\[ A(a)s = z \text{ if and only if } \langle a, s \rangle \rightarrow z \]

Theorem 3.2. A is a function

Given \( a \) and \( s \), there exists only one \( z \) such that \( \langle a, s \rangle \rightarrow z \).
\[\langle n, s \rangle \rightarrow n\]
\[\langle x, s \rangle \rightarrow s(\text{inl } x)\]
\[\langle a_1, s \rangle \rightarrow z_1 \quad \langle a_2, s \rangle \rightarrow z_2\]
\[\langle a_1 + a_2, s \rangle \rightarrow z_1 + z_2\]
\[\langle a_1 * a_2, s \rangle \rightarrow z_1 * z_2\]
\[\langle a_1 - a_2, s \rangle \rightarrow z_1 - z_2\]

Table 1: Semantics of arithmetic expressions

\[\langle \text{true}, s \rangle \rightarrow \text{true}\]
\[\langle \text{false}, s \rangle \rightarrow \text{false}\]
\[\langle x, s \rangle \rightarrow s(\text{inr } x)\]
\[A(a_1)s = z_1 \quad A(a_2)s = z_2\]
\[\langle a_1 = a_2, s \rangle \rightarrow z_1 = z_2\]
\[A(a_1)s = z_1 \quad A(a_2)s = z_2\]
\[\langle a_1 < a_2, s \rangle \rightarrow z_1 < z_2\]
\[\langle \neg b, s \rangle \rightarrow \neg t\]
\[\langle b_1, s \rangle \rightarrow t_1 \quad \langle b_2, s \rangle \rightarrow t_2\]
\[\langle b_1 \lor b_2, s \rangle \rightarrow t_1 \lor t_2\]
\[\langle b_1, s \rangle \rightarrow t_1 \quad \langle b_2, s \rangle \rightarrow t_2\]
\[\langle b_1 \land b_2, s \rangle \rightarrow t_1 \land t_2\]

Table 2: Semantics of boolean expressions
Proof. Structural Induction on a.

Definition 3.3. Semantics of Boolean Expressions

Table 2 presents the corresponding transition system

\(=, <, \neg, \lor, \land\) at the left denotes syntax and at the right the corresponding operations over the semantical domains.

We define the semantic function

\[ B(b)s = t \text{ if and only if } (b, s) \rightarrow t \]

Theorem 3.4. \(B\) is a function

Given \(b\) and \(s\), there exists only one \(t\) such that \((b, s) \rightarrow t\).

Proof. Structural Induction on \(b\).

4 Semantics of Statements

We use pattern matching and conditionals to specify the abstract machine. Conditionals supplement pattern matching allowing to impose conditions over pattern variables as well as to check the values of functions.

The way of reading our pattern matching definition is from top to bottom. The patterns are allowed to overlap (a variable overlaps all the patterns). When a pattern does not match, we continue with the following pattern.

Our pattern matching is exhaustive, this is achieved with variables in the pattern as the default case.

We specify a transition system with configurations of the form

\[ \langle S, n, y, E, \rho, \gamma, s \rangle. \]

\(S\) is the corresponding statement, \(n\) is the first counter, \(y\) the second counter (both \(n\) and \(y\) are natural numbers).
\(E\) is an environment of pairs \((\text{statement}, \text{counter})\) typed as:

\[ E : \text{Stack}(\text{Stm} \times N) \]
where *Stack* has constructors : and *nil*.

\( \rho \) is the first function that has as domain the set of *labels* and as codomain the set of *environments*.

\( \gamma \) is the second function used to know when a *while* statement is active.

We denote *States* by \( s \). We specify by \( s[x := n] \) the state such that \( (s[x := n] \cdot x) = n \) and \( (s[x := n] \cdot y) = (s \cdot y) \).

Table 3 presents the semantics of statements.

### 4.1 Use of the counters

We use the counters to number *while*-loops.

The first counter is a number associated to the statement when is being executed, the second counter equals the first natural number not assigned yet as first counter to a *while* statement.

When we have a *while* statement in the top of the environment occur one of the following two cases

1. the *while*-loop is active and was put in the stack to repeat the iteration. In this case its first counter does not change.

2. The *while* statement is not active. We assign to it as first counter the second counter.

We use the first counter also in relation to the statements *continue* and *break*.

When we find an active *while* statement, we have to execute its body and after repeat the execution of the statement, for this the body is put as the sentence at the top of the environment and the sentence *while* is put below it.

We associate as counter to the *while* statement in the environment the value of its first counter, and as counter for its body its successor. This will be used in relationship with statements *continue* and *break* and is explained afterwards when we explain the intended meaning of statements.

### 4.2 Use of the functions \( \rho \) and \( \gamma \)

The first function \( (\rho) \) stores for each *label*, an environment. When we meet a *goto label* statement, we continue the execution in the environment aso-
ciated to label.

The statement label: $S$ can be found "before" than a corresponding goto; in this case, when we reach the goto statement the function $\rho$ has an environment as definition for the corresponding label.

If this is not the case, the definition for the corresponding environment is done after finding the goto label statement. We go forward with the statements in the current environment until finding the statement labeled label. This statement and the ones in the current environment form the environment for label.

The second function ($\gamma$) is used to know if a while statement is active. Is defined in such a way that $\gamma_n = true$ if the while statement numbered $n$ is active and $\gamma_n = false$ otherwise.

### 4.3 The auxiliar function $h$

When we meet an if instruction, we choose the then or else part depending on the boolean condition. The corresponding sentence is put in the environment as first sentence to be executed.

We always put the next sentence in the top of the environment because in case of being a while instruction we give a special treatment (that can be seen in the management of the indexes).

Can be the case of a labeled statement appearing in the then or else part of an if statement. To define the corresponding environment in the function $\rho$ we define the function $h$, that appears at the end of Table 3.

We use the function $h$ also in case of a not active while statement. Since when an active while is find, this statement is put in the stack, any label found in the while’s body will have the while statement in its associated environment. In case the while statement is defused after, we have to associate to any label find in its body, an environment without this while statement. For this, we "compute" again the environment when we reach the defused while statement, applying function $h$. 
\[ \langle x \leftarrow a, n, y, E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip}, n, y, E, \rho, \gamma, s \mid \text{inl } \alpha \leftarrow \text{inl } A(a)s \rangle \]

\[ \langle x \leftarrow b, n, y, E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip}, n, y, E, \rho, \gamma, s \mid \text{inr } \beta \leftarrow \text{inr } B(b)s \rangle \]

\[ \langle \text{skip}, m, y, (S_1 ; S_2, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip}, m, y, (S_1, n) : (S_2, n) : E, \rho, \gamma, s \rangle \]

\[ \langle \text{skip}, m, y, (\text{while } b \text{ do } S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{while } b \text{ do } S, n, y, E, \rho, \gamma, s \rangle \quad \gamma n = \text{true} \]

\[ \langle \text{skip}, m, y, (\text{while } b \text{ do } S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{while } b \text{ do } S, y, s(y), E, \rho, \gamma, s \rangle \]

\[ \langle \text{skip}, m, y, (S, n) : E, \rho, \gamma, s \rangle \rightarrow (S, n, E, \rho, \gamma, s) \]

\[ \langle \text{skip}, n, y, \text{nil}, \rho, \gamma, s \rangle \rightarrow s \]

\[ \langle \text{label} : S, n, y, E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip}, n, y, (S, n) : E, \lambda z. \text{if } (z = \text{label}) \text{ then } (S, n) : E \text{ else } (\rho z), \gamma, s \rangle \]

\[ \langle \text{if } b \text{ then } S_1 \text{ else } S_2, n, y, E, \rho, \gamma, s \rangle \rightarrow^{b=\text{true}} \langle \text{skip}, n, y, (S_1, n) : E, h(S_1, n, E, h(S_2, n, E, \rho)), \gamma, s \rangle \]

\[ \langle \text{if } b \text{ then } S_1 \text{ else } S_2, n, y, E, \rho, \gamma, s \rangle \rightarrow^{b=\text{false}} \langle \text{skip}, n, y, (S_2, n) : E, h(S_1, n, E, h(S_2, n, E, \rho)), \gamma, s \rangle \]

\[ \text{Table 3 : Semantics of Statements} \]
4.4 The semantics and the intended meaning of statements

We show that the transition rules of the semantics of statements continue, break and goto follow their intended meaning.

4.4.1 The meaning of continue

If continue is inside an active while-loop has a counter of the form \( s(n) \). In this case \( \gamma n = true \). We have to skip all the sentences with the same number than continue (are sentences inside the same while-loop). We skip all these sentences until finding the while sentence with index \( n \). This is achieved by the following transitions:

\[
\langle \text{continue}, s(m), y, (S, s(n)) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{continue}, s(m), y, E, \rho, \gamma, s \rangle \quad \gamma m = true \land n = m
\]

\[
\langle \text{continue}, s(n), y, (\text{while } b \ \text{do } S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{while } b \ \text{do } S, n, y, E, \rho, \gamma, s \rangle \quad \gamma n = true
\]

If continue is at the greater level of the program (outside while loops) or inside a not active while loop its counter is 0, so the first two clauses does not match and the clause applied is one of the following

\[
\langle \text{continue}, m, y, (S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip}, m, y, (S, n) : E, \rho, \gamma, s \rangle
\]

\[
\langle \text{continue}, n, y, \text{nil}, \rho, \gamma, s \rangle \rightarrow s
\]

this is, behaves as skip, i.e. continues with the next sentence, and finishes with the execution in the case there are no more sentences in the program.

4.4.2 The meaning of break

If break is inside an active while-loop, has a counter of the form \( s(n) \) where \( \gamma n = true \). We have to skip all the sentences in the environment until finding the corresponding while sentence. This is achieved with the rules
\[
\langle \text{while } b \text{ do } S, n, y, \text{nil, } \rho, \gamma, s \rangle \rightarrow_{b=false}^{\text{false}} s
\]

\[
\langle \text{while } b \text{ do } S, m, y, E, \rho, \gamma, s \rangle \rightarrow_{b=false}^{\text{false}} \langle \text{skip, } m, E, h(S, 0, E, \rho), \lambda z. \text{if } (z = m) \text{ then false else } (\gamma z), s \rangle
\]

\[
\langle \text{while } b \text{ do } S, n, E, \rho, \gamma, s \rangle \rightarrow_{b=true}^{\text{true}} \langle \text{skip, } 0, (S, s(n)) : (\text{while } b \text{ do } S : n) : E, \rho, \lambda z. \text{if } (z = n) \text{ then true else } (\gamma z), s \rangle
\]

\[
\langle \text{continue, } m, y, (S, s(n)) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{continue, } m, y, E, \rho, \gamma, s \rangle \quad \gamma m = \text{true} \land n = m
\]

\[
\langle \text{continue, } s(n), y, (\text{while } b \text{ do } S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{while } b \text{ do } S, n, y, E, \rho, \gamma, s \rangle \quad \gamma n = \text{true}
\]

\[
\langle \text{continue, } m, y, E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip, } m, y, (S, n) : E, \rho, \gamma, s \rangle
\]

\[
\langle \text{continue, } n, y, \text{nil, } \rho, \gamma, s \rangle \rightarrow s
\]

\[
\langle \text{break, } m, y, (S, s(n)) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{break, } m, y, E, \rho, \gamma, s \rangle \quad \gamma m = \text{true} \land n = m
\]

\[
\langle \text{break, } s(n), y, (\text{while } b \text{ do } S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip, } n, y, E, \rho, \lambda z. \text{if } (z = n) \text{ then false else } (\gamma z), s \rangle \quad \gamma n = \text{true}
\]

\[
\langle \text{break, } m, y, (S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip, } n, y, (S, n) : E, \rho, \gamma, s \rangle
\]

\[
\langle \text{break, } n, y, \text{nil, } \rho, \gamma, s \rangle \rightarrow s
\]

Table 3: Semantics of Statements (continuation (a))
\[
\langle \textbf{goto} \text{ label}, m, y, E, \rho, \gamma, s \rangle \rightarrow \langle \textbf{skip}, m, y, (\rho \text{ label}), \rho, \gamma, s \rangle \quad (\rho \text{ label}) \neq \text{nil}
\]

\[
\begin{align*}
\langle \textbf{goto} \text{ label}, m, y, (S_1; S_2, n) : E, \rho, \gamma, s \rangle & \rightarrow \langle \textbf{goto} \text{ label}, m, y, (S_1, n) : (S_2, n) : E, \\
& \quad \rho, \\
& \quad \gamma, \\
& \quad s \rangle 
\end{align*}
\]

\[
\begin{align*}
\langle \textbf{goto} \text{ label}, m, y, (\text{label}' : S, n) : E, \rho, \gamma, s \rangle & \rightarrow \langle \textbf{goto} \text{ label}, m, y, E, \\
& \quad \lambda z. \text{if } (z = \text{label}') \text{ then } (S, n) : E \text{ else } (\rho \ z), \\
& \quad \gamma, \\
& \quad s \rangle
\end{align*}
\]

\[
\begin{align*}
\langle \textbf{goto} \text{ label}, m, y, (\text{label } S, n) : E, \rho, \gamma, s \rangle & \rightarrow \langle \text{skip}, n, y, \\
& \quad (S, n) : E, \\
& \quad \lambda z. \text{if } (z = \text{label}) \text{ then } (S, n) : E \text{ else } (\rho \ z), \\
& \quad \gamma, \\
& \quad s \rangle
\end{align*}
\]

\[
\begin{align*}
\langle \textbf{goto} \text{ label}, m, y, (\textbf{while } b \textbf{ do } S, n) : E, \rho, \gamma, s \rangle & \rightarrow \langle \textbf{goto} \text{ label}, m, y, E, \\
& \quad \rho, \\
& \quad \lambda z. \text{if } (z = n) \text{ then } \text{false } \text{ else } (\gamma \ z), \\
& \quad (\gamma \ n) = \text{true } \land s(n) \leq m
\end{align*}
\]

Table 3: Semantics of Statements (continuation (b))
\[ \langle \text{goto label}, m, y, (\text{while } b \text{ do } S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{goto label}, m, y, (S, 0) : E, \rho, \lambda z. \text{if } (z = n) \text{ then false else } (\gamma z), s \rangle \]

\[ \langle \text{goto label}, m, y, (\text{if } b \text{ then } S_1 \text{ else } S_2, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{goto label}, m, y, E, h(S_1, n, E, h(S_2, n, E, \rho), \gamma, s) \rangle \]

\[ \langle \text{goto label}, m, y, (S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{goto label}, m, y, E, \rho, \gamma, s \rangle \]

\[ h(label : S_1; S_2, n, E, \rho) = \lambda z. \text{if } (z = label) \text{ then } (S_1; S_2, n) : E \]
\[ \text{else } h(S_2, n, E, \rho) \]

\[ h(label : S, n, E, \rho) = \lambda z. \text{if } (z = label) \text{ then } (S, n) : E \]
\[ \text{else } \rho \]

\[ h(S_1; S_2, n, E, \rho) = h(S_2, n, E, \rho) \]

\[ h(S, n, E, \rho) = \rho \]

Table 3: Semantics of Statements (continuation (c))
\[ \langle \text{break}, s(m), y, (S, s(n)) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{break}, s(m), y, E, \rho, \gamma, s \rangle \quad \gamma m = \text{true} \land n = m \]

\[ \langle \text{break}, s(n), y, (\text{while } b \text{ do } S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip}, n, y, E, \rho, \lambda z. \text{if } (z = n) \text{ then false else } (\gamma z), \rho, \gamma, s \rangle \quad \gamma n = \text{true} \]

If \text{break} is at the greater level of the program (outside \text{while}-loops) or inside a not active \text{while} loop its counter is 0, so the first two clauses does not match and the clause applied is one of the following

\[ \langle \text{break}, m, y, (S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip}, m, y, (\rho \text{ label}) \rangle, \rho, \gamma, s \rangle \quad (\rho \text{ label}) \neq \text{nil} \]

\textbf{break} outside a \textbf{while}-loop behaves as skip.

The reason while we control in the transitions for \textbf{continue} and \textbf{break} if \( \gamma n = \text{true} \) is because we can find a sentence \textbf{continue} or \textbf{break} after a \text{jump} to a sentence inside a \textbf{while}-loop. The behaviour of the \textbf{continue} or \textbf{break} sentence depends on the \textbf{while} being active or not. In case is not active, we have not to skip sentences and we continue with the next statement below it.

\subsection{The meaning of goto}

If the sentence \textit{label} S is found before the \textbf{goto} \textit{label} sentence, the function \( \rho \) gives the environment whose sentences we have to execute. The corresponding transition is:

\[ \langle \text{goto label}, m, y, E, \rho, \gamma, s \rangle \rightarrow \langle \text{skip}, m, y, (\rho \text{ label}), \rho, \gamma, s \rangle \quad (\rho \text{ label}) \neq \text{nil} \]
When we find a labeled statement in the top of the environment with a different label than the \texttt{goto} statement, we redefine the function $\rho$ to answer the corresponding environment for the label.

\[
\langle \texttt{goto label}, m, y, (\texttt{label}': S, n) : E, \rho, \gamma, s \rangle \rightarrow \\
\langle \texttt{goto label}, \\
m, \\
y, \\
E, \\
\lambda z. \text{if } (z = \texttt{label}') \text{ then } (S, n) : E \text{ else } (\rho z), \\
\gamma, \\
s \rangle
\]

When we find the statement $\texttt{label} : S$ on top of the stack our transition is

\[
\langle \texttt{goto label}, m, y, (\texttt{label} S, n) : E, \rho, \gamma, s \rangle \rightarrow \\
\langle \texttt{skip}, \\
n, \\
y, \\
(S, n) : E, \\
\lambda z. \text{if } (z = \texttt{label}) \text{ then } (S, n) : E \text{ else } (\rho z), \\
\gamma, \\
s \rangle
\]

If the sentence $\texttt{goto label}$ is inside an active \texttt{while}-loop and $\texttt{label} S$ is found below the \texttt{while}-loop we have to jump outside the \texttt{while}-loop. This is achieved with the transition

\[
\langle \texttt{goto label}, m, y, (\texttt{while } b \text{ do } S, n) : E, \rho, \gamma, s \rangle \rightarrow \\
\langle \texttt{goto label}, \\
s(n), \\
y, \\
E, \\
\rho, \\
\lambda z. \text{if } (z = n) \text{ then } \texttt{false} \text{ else } (\gamma z), \\
(\gamma n) = \texttt{true} \land s(n) \leq m \rangle
\]
Can be the case that label $S$ appears after the $\texttt{goto label}$ inside a non-active \texttt{while}-loop, in this case we search label $S$ in the body of the \texttt{while}-loop.

$$\langle\texttt{goto label}, m, y, (\texttt{while } b \texttt{ do } S, n) : E, \rho, \gamma, s \rangle \rightarrow$$

$$\langle\texttt{goto label}, m, y, E, h(S, 0, E, \rho), \lambda z. \text{if } (z = n) \text{ then false else } (\gamma z), s \rangle$$

Label $S$ can appear inside an $\texttt{if}$ instruction:

$$\langle\texttt{goto label}, m, y, (\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, n) : E, \rho, \gamma, s \rangle \rightarrow$$

$$\langle\texttt{goto label}, m, y, E, h(S_1, n, E, h(S_2, n, E, \rho)), \gamma, s \rangle$$

In any other case, we go ahead searching a statement of the form label $S$:

$$\langle\texttt{goto label}, m, y, (S, n) : E, \rho, \gamma, s \rangle \rightarrow \langle\texttt{goto label}, m, y, E, \rho, \gamma, s \rangle$$

To end, we define the semantic partial function

$S : \texttt{State} \rightarrow \texttt{State}$ by

$S(S)s = s'$ if and only if $\text{initial}(S)s = \langle\texttt{skip}, 0, 0, \alpha, \beta, s \rangle \rightarrow^* s'$

where

$\alpha = \lambda z. \text{nil}$ and $\beta = \lambda z. \text{false}$
5 Properties of the semantics

Definition 5.1. Intermediate and final configurations

Our transitions have the form

\[ \langle S, n, y, E, \rho, \gamma, s \rangle \rightarrow \delta \]

where \( \delta \) has the form \( \langle S', n', y', E', \rho', \gamma', s' \rangle \) or the form \( s' \). The first are called \textit{intermediate configurations}, and the last \textit{final configurations}.

We say \( \langle S, n, y, E, \rho, \gamma, s \rangle \) is \textit{stuck} is there is no \( \delta \) such that \( \langle S, n, y, E, \rho, \gamma, s \rangle \rightarrow \delta \).

Definition 5.2. Derivation Sequence, termination, loops

A derivation sequence is

1. a finite sequence of configurations

\[ \delta_0, \delta_1, \ldots, \delta_k \]

where \( \delta_i \rightarrow \delta_{i+1} \) for \( 0 \leq i < k \) and where \( \delta_k \) is a final configuration or a stuck configuration.

2. an infinite sequence

\[ \delta_0, \delta_1, \ldots \]

where \( \delta_i \rightarrow \delta_{i+1} \) for \( 0 \leq i \).

We say that a statement \( S \) with counters \( n \) and \( y \), environment \( E \), functions \( \alpha \) and \( \beta \) and state \( s \), \textit{terminates} if and only if there is a finite derivation sequence starting in \( \langle S, n, y, E, \alpha, \beta, s \rangle \) and \textit{loops} if and only if there is a infinite derivation sequence starting in \( \langle S, n, y, E, \alpha, \beta, s \rangle \).

We say that a statement \( S \) from an state \( s \), terminates if \( \text{initial}(S)s \) terminates and loops if \( \text{initial}(S)s \) loops.

Theorem 5.3. There are no stuck configurations

\[ \square \]
Theorem 5.4. Determinism

There is exactly one derivation sequence, starting in a configuration $\delta$, i.e. if $\delta \rightarrow \delta'$ and $\delta \rightarrow \delta''$ then $\delta' = \delta''$.

Proof. There is only one possible transition for each possible pattern and condition. \qed

Corollary 5.5. A statement $S$ from an state $s$, terminates or loops.

Proof. If $S$ terminates, exists $k$ and $s'$ such that $\text{initial}(S)s \rightarrow^k s'$. If $S$ from $s$ loops, has an infinite derivation sequence. To both terminate and loop contradicts determinism. \qed

6 Related Work

The work in [2] presents a formal semantics based in evolving algebras. There treatment of break, continue and goto is to call a function \textit{NextTask} that is primitive, static and belongs to one of the algebras, so is not specified its implementation.

The treatment in [5, 6] is rather different to our work, we base our definition in the index, while Norris defines special statement values (BreakVal, ContVal, RetVal and StmtVal) to whom occur transitions when a statement break, continue, return or an ordinary evaluation termination happens respectively.

He’s semantics is big step and does not define the semantic of goto.

7 Conclusions

We have presented the semantics for break, continue and goto using an stack machine, with two counters and three functions.

It is known that we can express the same programs without using break, continue nor goto. In the present paper we have tried to show that the introduction of such constructions in giving program semantics is not so complicated nor so obscure as one could imagine before hand.
References


