Analysis of Loop Semantics using *S*-Formulas

Aleksandar Kupusinac [1], Dusan Malbaski [1]

[1]*Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia*

***Abstract* – There are three possible behavioral patterns for the *WHILE* loop: it does not terminate, it potentially terminates and its termination is guaranteed. Based on that, to describe the behavior of the *WHILE* loop we introduce appropriate formulas of the first-order predicate logic defined on the abstract state space (briefly *S*-formulas). This paper presents our approach to analyzing the *WHILE* loop semantics that is solely based on the first order predicate logic.**

***Keywords* – Theory of programming, loop semantics.**

1. Introduction

Semantics reveals the meaning of syntactically valid strings in a language. For natural languages, this means correlating sentences and phrases with the objects, thoughts, and feelings of our experiences. For programming languages, semantics describes the behavior that a computer follows when executing a program in the language. We might disclose this behavior by describing the relationship between the input and output of a program or by a step-by-step explanation of how a program will execute on a real or an abstract machine [1].

In giving a formal semantics to a programming language we are concerned with building a mathematical model. Its purpose is to serve as a basis for understanding and reasoning about how programs behave. Not only is a mathematical model useful for various kinds of analysis and verification, but also, at a more fundamental level, because simply the activity of trying to define the meaning of program constructions precisely can reveal all kinds of subtleties of which it is important to be aware. The techniques used in semantics lean heavily on mathematical logic. They are not always easily accessible to a programmer, without a good background in logic [2].

The relevance of conditions in reasoning about programs was known to von Neumann and Turing [3]. Floyd suggested that the assertions encapsulate the meaning of a program [4]. Hoare introduced an axiomatic approach, well known as Hoare logic, where the laws of reasoning with assertions were accepted as an axiomatic definition of the meaning of the whole programming language [5]. The axiomatic approach was adopted also by Dijkstra [6,7]. After that, Hoare and Jifeng in *Unifying Theories of Programming* introduced an approach that is not axiomatic [8]. They proved the necessary laws as theorems based on an independent mathematical definition of the meaning of a program as a relation. The axioms postulated in Hoare logic are proved as the theorems of the more basic theory, such as the first-order predicate logic.

In this paper, we consider *WHILE* loop semantics by using so-called *S*-formulas, that are actually formulas of the first-order predicate logic defined on the abstract state space. That actually was the reason for naming them"*S*-formulas", after the word "state". Thus, our approach is based on the idea that programs may be treated as predicates [8,9,10,11,12] and clearly separates the interpretation domain from the abstract state domain [13]. Using a set of abstract states provides an advantage to our approach. This means that we do not need an exact description of every abstract state, thus avoiding the use of the program state vector (vector of all program variables). It is known that the use of state vector introduces certain difficulties [7].

The paper is structured as follows. The Section 2 considers behavioral patterns for the *WHILE* loop. In the Section 3, we introduce appropriate *S*-formulas used for analyzing *WHILE* loop semantics. Based on that, we will analyze concrete examples in the Section 4. The paper ends with conclusions and directions for further research.

2. Behavioral patterns of the *WHILE* loop

In programming languages, a *WHILE* loop is a statement that means the cyclic execution of the loop body based on a given Boolean condition. In every cycle, the condition is checked before the execution of the loop body. If the condition is true, the loop body will be executed and this is repeated until the condition becomes false. If the condition becomes false after the finite number of cycles, we say that the given *WHILE* loop terminates.

Let us consider the following *WHILE* loop written in pseudo code:

```
WHILE a>0 DO
BEGIN
      IF a>5 THEN
            a:=a+1;
      a:=a-1;
END;
```

Obviously, for $a>5$ the *WHILE* loop does not terminate. The *WHILE* loop terminates for $a\leq 5$, but we note three cases:

- for $a\leq 0$, the loop body is never executed,
- for $a=1$, the loop body is executed only once,
- for $2\leq a\leq 5$, the loop body is executed several times.

Now, let us consider the second example:

```
WHILE a>0 DO
BEGIN
      IF a>5 THEN
            a:=a+1;
      a:=a-1 ∨ a+1;
END;
```

Again, the *WHILE* loop terminates for $a\leq 0$ since its body is never executed. But, for $1\leq a\leq 5$ the *WHILE* loop may or may not terminate, i.e. it terminates potentially, since

```
a:=a-1∨a+1;
```

is a non-deterministic statement. Let us note that for $1\leq a\leq 5$ the previous *WHILE* loop surely terminates, i.e. the termination is guaranteed. Based on that, we observe three behavioral patterns of the *WHILE* loop:

- non-termination,
- potential termination,
- guaranteed termination.

3. *S*-formulas

Formulas of the first-order predicate logic defined on the abstract state space we call briefly *S*-formulas. In this paper we use the following concepts and notation:

- The set of abstract states A,
- State variables (*S*-variables) $x, y, z, \ldots$,
- State constants (*S*-constants) $s_1, s_2, s_3, \ldots$,
- Unary *S*-formulas or *S*-predicates $P, Q, B, \ldots$,
- Binary *S*-formulas or *S*-relations $S_1, S_2, S_3, \ldots$,
- Program variables $a, b, c, \ldots$,
- Program constants $c_1, c_2, c_3, \ldots$.

Let $\{a_1, a_2, \ldots, a_n\}$ be a set of program variables, which take values from sets $D_1, D_2, \ldots, D_n$ respectively. Interpretation of the set A with respect to the set $\{a_1, a_2, \ldots, a_n\}$ is a bijection that maps any *S*-constant from A to the appropriate vector of program constants from $D_1, D_2, \ldots, D_n$ (usually called state vector). *S*-relation $S(x,y)$ contains ordered pairs (x,y), where $x\in A$ is the initial state and $y\in A$ is the final state. Interpreted *S*-relation on the set A is called syntactic unit on program variables $\{a_1, a_2, \ldots, a_n\}$. A syntactic unit may be written in many different ways (program code is just one of them), and it can refer to a statement, block, subprogram or program. This means that we observe two domains: the abstract state domain with *S*-constants, *S*-variables, *S*-predicates and *S*-relations and the interpretation domain with vectors of program constants, program variables, predicates and syntactic units. To simplify, *S*-constant is interpreted as a vector of program constants from the set $D_1, D_2, \ldots, D_n$, *S*-predicate is interpreted as a Boolean expression, and *S*-relation as a syntactic unit with program variables $\{a_1, a_2, \ldots, a_n\}$. Interpretation is denoted by ":". For example, x: $a>0\wedge b=5$ means that *S*-variable x represents all states in which program variables a and b satisfy $a>0$ and $b=5$.

Let us consider the following *WHILE* loop:

WHILE condition DO
loop_body;

Let
B: condition,
S: loop_body,
$x, y, y_1, y_2, \ldots, y_n \in A$.

S-formulas S_P and S_G are defined in the following way:

DEFINITION 3.1.
$\forall x\forall y\ S_P(x,y) \Leftrightarrow [\neg B(x) \wedge x=y] \vee [B(x) \wedge S(x,y) \wedge \neg B(y)] \vee [\exists y_1\exists y_2 \ldots \exists y_n\ B(x) \wedge S(x,y_1) \wedge B(y_1) \wedge S(y_1,y_2) \wedge B(y_2) \wedge S(y_2,y_3) \wedge \ldots \wedge B(y_n) \wedge S(y_n,y) \wedge \neg B(y)]$,

DEFINITION 3.2.
$\forall x\forall y\ S_G(x,y) \Leftrightarrow [\neg B(x) \wedge x=y] \vee [B(x) \wedge S(x,y) \wedge \neg B(y)] \vee [\forall y_1\forall y_2 \ldots \forall y_n\ B(x) \wedge S(x,y_1) \wedge B(y_1) \wedge S(y_1,y_2) \wedge B(y_2) \wedge S(y_2,y_3) \wedge \ldots \wedge B(y_n) \wedge S(y_n,y) \wedge \neg B(y)]$.

If we analyze the given *WHILE* loop using both formulas we arrive to the more complete insight in its behavior:
1.) If the initial state x satisfies $\forall y$, $(x,y)\notin S_P$ then the loop does not terminate from the state x.
2.) If the initial state x satisfies $\forall y$, $(x,y)\in S_P \wedge (x,y) \notin S_G$ then the loop potentially terminates from the state x.
3.) If the initial state x satisfies $\forall y$, $(x,y)\in S_G$ then the loop inevitably terminates from the state x.

S-formulas S_P and S_G consist of three disjunctive parts. The first $\forall x\forall y\ \neg B(x) \wedge x=y$ covers the case

when the loop body is not executed at all. The second $\forall x \forall y\ B(x) \wedge S(x,y) \wedge \neg B(y)$ refers to the case when the body is executed exactly once. The third part is an $(n+2)$-ary S-formula where n is a number of intermediate states $y_1, y_2, \ldots, y_n$ traced by the loop. The difference between S-formulas S_P and S_G is in the third part, in which S_P contains existential quantifiers (i.e. $\exists y_1 \exists y_2 \ldots \exists y_n$) while in S_G the quantifiers are universal (i.e. $\forall y_1 \forall y_2 \ldots \forall y_n$). Apparently, the S-formula S_G is stronger than S_P, i.e. $S_G \subseteq S_P$, or $\forall x \forall y\ S_G(x,y) \Rightarrow S_P(x,y)$. In other words, if $(x,y) \in S_G$, then $(x,y) \in S_P$, while the opposite is not true. The S-formula S_G eliminates all initial states from which the loop (only) potentially terminates thus leading to the guaranteed termination.

Potential termination and non-termination are more of a theoretical value, while for the programming practice the most important case is guaranteed termination. Moreover, loops must terminate in the acceptable time interval [14,15]. In the Example 4.1 we will show that an infinite cycle conforms to the empty set. In the Example 4.4 we will consider a potential termination, where the difference between the S-formulas S_P and S_G is apparent.

4. Examples

4.1. Let us prove $\forall x \forall y\ S_P(x,y) \equiv \bot$ for the infinite loop

```
WHILE true DO
      loop_body;
```

Let
B: `true`
S: `loop_body`
$x, y, y_1, y_2, \ldots, y_{k-1}, y_k, y_{k+1} \in A$

Clearly,
$\forall x\ B(x) \equiv \top$.

Let
$\forall x \forall y_1\ S(x,y_1)$
$\forall y_1 \forall y_2\ S(y_1,y_2)$
. . .
$\forall y_{k-1} \forall y_k\ S(y_{k-1},y_k)$
$\forall y_k \forall y_{k+1}\ S(y_k,y_{k+1})$
. . .

We will prove by induction that the program does not terminate from the initial state x. After the first pass $B(y_1)$ is true, so apparently the program does not terminate. If we suppose that the program did not terminate after the k-th pass and $B(y_k)$ is true, then after the k+1-th pass $B(y_{k+1})$ holds. Consequently, the program does not terminate after the k+1-th pass. From the Definition 3.1 we conclude that S-formula S_P does not contain the ordered pair (x,y) with x and y being respectively the initial and the final state. This proves that for any initial state x a final state does not exist, i.e. that S_P is an empty set or $\forall x \forall y\ S_P(x,y) \equiv \bot$.

4.2. Let us prove $\forall x \forall y\ S_G(x,y) \equiv x{:}\ a \leq 0 \wedge y{=}x$ for

```
WHILE a>0 DO
      a:=a+1;
```

Let
B: `a>0`
S: `a:=a+1;`
$x, y, y_1, y_2, \ldots, y_{k-1}, y_k, y_{k+1} \in A$

Apparently,
$\forall x,\ x{:}\ a{>}0,\ B(x) \equiv \top$,
$\forall x,\ x{:}\ a \leq 0,\ \neg B(x) \equiv \top$.

From the Definition 3.2, we obtain:
$\forall x,\ x{:}\ a \leq 0,\ \forall y\ S_G(x,y) \Leftrightarrow \neg B(x) \wedge x{=}y \Leftrightarrow \top \wedge x{=}y \Leftrightarrow x{=}y$,
i.e.
$\forall x \forall y\ S_G(x,y) \Leftrightarrow x{:}\ a \leq 0 \wedge y{=}x$.

By a', a'', ... , $a^{(k)}$ we denote the value of the program variable a after the first, second, ... , k-th pass. S-relation S is the assignment, so we obtain:

$\forall x \forall y_1\ S(x,y_1) \Leftrightarrow x{:}\ a{>}0 \wedge y_1{:}\ a'{=}a{+}1$
$\forall y_1 \forall y_2\ S(y_1,y_2) \Leftrightarrow y_1{:}\ a'{>}0 \wedge y_2{:}\ a''{=}a'{+}1$
. . .
$\forall y_{k-1} \forall y_k\ S(y_{k-1},y_k) \Leftrightarrow y_{k-1}{:}\ a^{(k-1)}{>}0 \wedge y_k{:}\ a^{(k)}{=}a^{(k-1)}{+}1$
$\forall y_k \forall y_{k+1}\ S(y_k,y_{k+1}) \Leftrightarrow y_k{:}\ a^{(k)}{>}0 \wedge y_{k+1}{:}\ a^{(k+1)}{=}a^{(k)}{+}1$
. . .

The proof that the program does not terminate from the initial state $x{:}\ a{>}0$ is by induction. Since $B(y_1)$ is true, the program does not terminate after the first pass. If we suppose that the program did not terminate after the k-th pass and $B(y_k)$ is true, then after the k+1-th pass $B(y_{k+1})$ holds, so the program does not terminate after the k+1-th pass. From the Definition 3.1 (similarly to the Example 4.1) we obtain

$\forall x,\ x{:}\ a{>}0,\ \forall y\ S_P(x,y) \equiv \bot$.

Since $\mathcal{S}_G \subseteq \mathcal{S}_P$, we conclude
$\forall x,\ x{:}\ a{>}0,\ \forall y\ S_G(x,y) \equiv \bot$.

So far we have proven that starting from the state x: $a\leq 0$ the cycle is guaranteed to terminate, while from the state x: $a>0$ it does not terminate. Finally, we obtain

$\forall x \forall y\ S_G(x,y) \equiv [x: a\leq 0 \wedge y=x] \vee \bot$,

i.e.

$\forall x \forall y\ S_G(x,y) \equiv x: a\leq 0 \wedge y=x$.

4.3. In this example we will prove $\forall x \forall y\ S_G(x,y) \Leftrightarrow [x: a=0 \wedge y: a=1] \vee [x: a\neq 0 \wedge y=x]$ for

```
WHILE a=0 DO
      a:=a+1;
```

Let

B: `a=0`

S: `a:=a+1;`

$x, y \in A$

Clearly,

$\forall x, x: a=0, B(x) \equiv \top$,

$\forall x, x: a\neq 0, \neg B(x) \equiv \top$.

S-relation is the assignment and we obtain:

$\forall x, x: a=0, \forall y\ S(x,y) \Leftrightarrow x: a=0 \wedge y: a=1$.

Definition 3.2 leads to

$\forall x, x: a=0, \forall y\ S_G(x,y) \Leftrightarrow \neg B(x) \wedge \neg B(y) \wedge S(x,y) \Leftrightarrow \top \wedge \top \wedge [x: a=0 \wedge y: a=1]$,

i.e.

$\forall x \forall y\ S_G(x,y) \Leftrightarrow x: a=0 \wedge y: a=1$.

On the other hand

$\forall x, x: a\neq 0, \forall y\ S_G(x,y) \Leftrightarrow \neg B(x) \wedge \text{x=y} \Leftrightarrow \top \wedge [x: a\neq 0 \wedge y=x]$,

i.e.

$\forall x \forall y\ S_G(x,y) \Leftrightarrow x: a\neq 0 \wedge y=x$.

We have proven that starting from any state $x \in A$ the cycle is guaranteed to terminate. Finally,

$\forall x \forall y\ S_G(x,y) \Leftrightarrow [x: a=0 \wedge y: a=1] \vee [x: a\neq 0 \wedge y=x]$.

4.4. Prove that $\forall x \forall y\ S_G(x,y) \Leftrightarrow x: a\leq 0 \wedge y=x$ for

```
WHILE a>0 DO
      a:=a-1∨a+1;
```

Let

B: `a>0`

S: `a:=a-1∨a+1;`

$x, y, y_1, y_2, \ldots, y_{k-1}, y_k, y_{k+1} \in A$

Obviously,

$\forall x, x: a>0, B(x) \equiv \top$,

$\forall x, x: a\leq 0, \neg B(x) \equiv \top$.

From the Definition 3.2 we obtain:

$\forall x, x: a\leq 0, \forall y\ S_G(x,y) \Leftrightarrow \neg B(x) \wedge x=y \Leftrightarrow \top \wedge x=y \Leftrightarrow x=y$,

i.e.

$\forall x \forall y\ S_G(x,y) \Leftrightarrow x: a\leq 0 \wedge y=x$.

S-relation S is a non-deterministic syntactic unit, so we obtain:

$\forall x, x: a>0, \forall y\ S(x,y) \Leftrightarrow x: a>0 \wedge y: a'=a-1 \vee a'=a+1$.

Based on that, we obtain the following S-formulas:

$\forall x \forall y_1\ S(x,y_1) \Leftrightarrow x: a>0 \wedge y_1: a'=a-1 \vee a'=a+1$

$\forall y_1 \forall y_2\ S(y_1,y_2) \Leftrightarrow y_1: a'>0 \wedge y_2: a''=a'-1 \vee a''=a'+1$

. . .

$\forall y_{k-1} \forall y_k\ S(y_{k-1},y_k) \Leftrightarrow y_{k-1}: a^{(k-1)}>0 \wedge y_k: a^{(k)}=a^{(k-1)}-1 \vee a^{(k)}=a^{(k-1)}+1$

$\forall y_k \forall y_{k+1}\ S(y_k,y_{k+1}) \Leftrightarrow y_k: a^{(k)}>0 \wedge y_{k+1}: a^{(k+1)}=a^{(k)}-1 \vee a^{(k+1)}=a^{(k)}+1$

. . .

We will prove by induction that the program potentially terminates from the initial state x: $a>0$. After the first pass $B(y_1)$ may or may not hold so the program may or may not terminate. If we suppose that the program did not terminate after the k-th pass, then after the k+1-th pass $B(y_{k+1})$ may or may not hold. Apparently, the program may or may not terminate, so we conclude that starting from the state x: $a>0$ program potentially terminates. In other words, based on the Definition 3.1 we obtain:

$\forall x, x: a>0, \forall y\ S_P(x,y) \equiv \top$.

However, from the Definition 3.2 follows:

$\forall x, x: a>0, \forall y\ S_G(x,y) \equiv \bot$.

Finally, we obtain:

$\forall x \forall y\ S_G(x,y) \Leftrightarrow [x: a\leq 0 \wedge y=x] \vee \bot$,

i.e.

$\forall x \forall y\ S_G(x,y) \Leftrightarrow x: a\leq 0 \wedge y=x$.

4.5. We will prove $\forall x \forall y\ S_G(x,y) \Leftrightarrow [x: a\leq 0 \wedge y=x] \vee [x: a>0 \wedge y: a=0]$ for

```
WHILE a>0 DO
BEGIN
      a:=a-1;
      b:=b-1∨b+1;
END;
```

Let

B: `a>0`

S: `a:=a-1; b:=b-1`$\vee$`b+1;`
$x, y, y_1, y_2, \ldots, y_{k-1}, y_k, y_{k+1} \in A$

As in the Example 4.4, based on the Definition 3.2 we obtain:

$\forall x, x: a \leq 0, \forall y\ S_G(x,y) \Leftrightarrow \neg B(x) \wedge x=y \Leftrightarrow \top \wedge x=y \Leftrightarrow x=y,$
i.e.
$\forall x \forall y\ S_G(x,y) \Leftrightarrow x: a \leq 0 \wedge y=x.$

In addition, we obtain the following S-formulas:

$\forall x \forall y_1\ S(x,y_1) \Leftrightarrow x: a>0 \wedge y_1: a'=a-1 \wedge (b'=b-1 \vee b'=b+1)$
$\forall y_1 \forall y_2\ S(y_1,y_2) \Leftrightarrow y_1: a'>0 \wedge y_2: a''=a'-1 \wedge (b''=b'-1 \vee b''=b'+1)$
. . .
$\forall y_{k-1} \forall y_k\ S(y_{k-1},y_k) \Leftrightarrow y_{k-1}: a^{(k-1)}>0 \wedge y_k: a^{(k)}=a^{(k-1)}-1 \wedge (b^{(k)}=b^{(k-1)}-1 \vee b^{(k)}=b^{(k-1)}+1)$
$\forall y_k \forall y_{k+1}\ S(y_k,y_{k+1}) \Leftrightarrow y_k: a^{(k)}>0 \wedge y_{k+1}: a^{(k+1)}=a^{(k)}-1 \wedge (b^{(k+1)}=b^{(k)}-1 \vee b^{(k+1)}=b^{(k)}+1)$
. . .

From the Definition 3.2 follows:
$\forall x, x: a>0, \forall y, y: a=0, S_G(x,y),$
i.e.
$\forall x \forall y\ S_G(x,y) \Leftrightarrow x: a>0 \wedge y: a=0.$

We have proven that starting from any state $x \in A$ cycle must terminate. Finally, we obtain
$\forall x \forall y\ S_G(x,y) \Leftrightarrow [x: a \leq 0 \wedge y=x] \vee [x: a>0 \wedge y: a=0].$

5. Conclusion

In this paper we have introduced appropriate S-formulas that describe three possible behavioral patterns for the *WHILE* loop: it does not terminate, it potentially terminates and its termination is guaranteed. Based on that, we have developed an approach to the analysis of loop semantics and considered several examples. The future work will be aimed towards investigating algorithms for automated loop semantics proofs, thus providing a practical aspect to this research.

Acknowledgements

This work was partially supported by the Ministry of Science and Education of the Republic of Serbia within the projects: ON 174026 and III 044006.

References

[1]. Slonneger, K. and Kurtz, B. L.: Formal Syntax and Semantics of Programming Languages. Addison-Wesley Publishing Company, Inc., 1995.
[2]. Winskel, G.: The formal semantics of programming languages : an introduction, (2nd printing). The MIT Press, 1994.
[3]. Turing, A. M.: Checking a large routine. Report of a Conference on High Speed Automatic Calculating Machines, University Mathematical Laboratory, Cambridge, 67–69, 1949.
[4]. Floyd, R. W.: Assigning meanings to programs. In Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics. American Mathematical Society, Providence, RI, USA, 19-32, 1967.
[5]. Hoare, C. A. R.: An Axiomatic Basis for Computer Programming. Comunications of the ACM, 12(10), 576-585, October 1969.
[6]. Dijkstra, E. W.: Guarded Commands, Nondeterminacy and Formal Derivation of Programs. Comunications of the ACM, 18(8), 453-457, August 1975.
[7]. Dijkstra, E. W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs, NJ, USA, 1976.
[8]. Hoare, C. A. R. and Jifeng, H.: Unifying Theories of Programming. Prentice-Hall, London, 1998.
[9]. Pratt, V. R.: Semantical Considerations on Floyd-Hoare logic. Tech. Rep. MIT/LCS/TR 168, Massachusetts Institute of Technology, Laboratory for Computer Science, Massachusetts, USA, 1976.
[10]. Hoare, C. A. R. and Roscoe, A. W.: Programs as executable predicates. In Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'84). Institute for New Generation Computer Technology (ICOT, Ed.), Tokyo, Japan, 220-228, 1984.
[11]. Zwiers, J. and Roever, W.: Predicates are predicate transformers: a unified compositional theory for concurrency. In Proceedings of the eighth annual ACM Symposium on Principles of distributed computing (PODC'89, Edmonton, Alberta, Canada). ACM, New York, NY, USA, 265-279, 1989.
[12]. Hoare, C. A. R.: Programs are predicates. In Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'92). Institute for New Generation Computer Technology (ICOT, Ed.), Tokyo, Japan, 211-218, 1992.
[13]. Nakano, M., Ogata, K., Nakamura, M. and Futatsugi, K.: Automating invariant verification of behavioral specifications. In Proceedings of the Sixth International Conference on Quality Software (QSIC'06). IEEE Computer Society, Beijing, China, 49-56, 2006.
[14]. Hehner, E. C. R.: Specifications, Programs, and Total Correctness. Science of Computer Programming, 34, 191-205, 1999.
[15]. Hehner, E. C. R.: A Practical Theory of Programming. University of Toronto, 2011. www.cs.utoronto.ca/~hehner /aPToP

Corresponding author: Aleksandar Kupusinac
Institution: University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia