Circular Logic for Computational Semantics

Dr. Axel Arturo Barceló Aspeitia

Instituto de Investigaciones Filosóficas
Universidad Nacional Autónoma de México
abarcelo@minerva.filosoficas.unam.mx

Abstract. This article presents an expressively rich circular logic for computational semantics based on Barwise and Moss [7]. The main advantage of this logic over first-order well-founded ones is that a single formula of the language can trace the change of values continuously and at different memory states. In order to account for circularity, the system uses systems of equations. The formulae of this logic are defined as the unique solutions to systems of equations. The equations that constitute the systems are of the form 'x = t', where t is a term and x is a term variable. The terms in this language are syntactically similar to formulae in traditional first order logic except for the occurrence of clock terms, which account for time in the performance of computational programs. This paper introduces the systems of equations syntax and offers a semantic interpretation on streams. Then, uses this language to produce a logic for the standard connectives defined over binary streams. The logic here presented is both sound and complete. The paper finishes pointing out several further developments necessary to take full advantage of the logic here presented.

1 Introduction

In mathematical logic, the semantic interpretation of a formal language consists of a consequence relation among well-formed formulae and a valuation function mapping well-formed phrases into mathematical entities of an appropriate domain. These mathematical entities play the role of intended meanings in the semantics. In the case of semantics for programming languages, the intended meanings are computational behaviors (operations and/or changes in their data structures). The challenge for computer semanticists has been to find the adequate domain of mathematical entities for better modeling such behavior. Traditional semantics for mathematical logic borrow their models from the field of set theory. When applied to programming languages, this commonly results in denotational semantics where the interpretations of statements in the programming language are functions defined over states and data values. The resulting semantics model informational change inside the computer as series of discrete changes from one single value to another at a single memory unit. By avoiding the constraints of classical logic and its semantics, it is possible to

obtain more powerful tools for the modeling of computational behavior. In particular, it is possible to use a non-classical variant of circular logic to model informational change in a continuous manner, tracing the data changes in several memory units at once. The goal of this paper is precisely to provide such a logic.

2 Syntax

Because of its relative youth, circular logic is still at a developing stage. Since it evolved out of non-well-founded set theory, its mayor developments have been in model theory. In contrast, its proof theory does not even have a standard syntax. There have been several developments in the area (by myself and others), none of which has become widespread enough to become customary. I will present here a new proposal. This proposal is based on a mechanism devised by Barwise and Moss [7] for presenting and identifying non-well-founded sets, also known as hypersets.

The basic intuition behind my proposal is that logical formulae can be identified as the unique solution of systems of equations. For example, $(p \wedge \neg q)$ is the unique solution to systems of equations such as $\{ x = (p \wedge y), y = \neg q \}$, $\{ x = (y \wedge z), y = p, z = \neg w, w = q \}$, or simply $\{ x = (p \wedge \neg q) \}$. In traditional well-founded logic, the proposal to express formulae in terms of systems of equations would seem trifling. However, it proves very useful in circular logic. Consider, for example, the system of equations $\{ x = (y \wedge z), y = p, z = \neg x \}$. As such, it has no solution in well-founded logic. However, the solution lemma, a consequence of the anti-foundation axiom, tells us that there must be a unique solution to this system of equations. The following proposal exploits this mathematical fact to provide a language explicitly designed for semantically modeling circularity.

The only extra element added to the syntax is the use of square brackets. They serve to avoid vicious circularity. Brackets, in this language, work as *clock* items. Their role is to account for time and the direction of information flow within a circuit. Information within the circuit flows from its non-bracketed portion to the bracketed one. The valuation of the bracketed portion of a formula depends on the information contained in the non-bracketed portion. In consequence, it has to be valuated an instant later than the non-bracketed one.

2.1 Language

Let **O** be a set of *operation* symbols of the form O^n_i where *n* is the n-arity of the operation. Let **S**, **X** and **C** be distinct sets of symbols, all of them different from **O**. The symbols in **S** perform as *input* or *memory* symbols. The symbols in **X** perform as *term variables*, while those in **C** serve as *initial state constants*. For convenience, use Roman lower case letters *a*, *b*, *c*, . . . for initial state constants, *p*, *q*, *r*, . . . for input symbols, and *z*, *y*, *x*, . . . for term variables. Let Σ = **S** O **X** **C** {'=', '[', ']'} be the alphabet of our language.

Definitions 2.1.1

As usual, the operation letters applied to input and variable symbols generate the *terms*:

1. Variables and input letters are terms.
2. If O^n_k is an operation letter and $t_1, t_2, \ldots t_n$ are terms, then $O^n_k (t_1, t_2, \ldots t_n)$ is a term.
3. If t is a term and *a* is an initial state constant, [*a*, t] is a term, too.
4. An expression is a term only if it can be shown to be a term on the basis of conditions 1, 2 and 3.

An *equation* is an expression of the form x = t, where x is a term variable and t is a term. A *(flat) system* of equations is an ordered pair of an equation and a finite, possibly empty, set of equations in a language. The first equation is called the *main* equation of the system.

Defintions 2.1.2

Given a term t_2 of the form [*a*, t_1], t_1 and whatever occurs in t_1 is said to occur *delayed* in t_2.

Given a system of equations, a variable x is said to *directly depend* on another variable y iff there is an equation x = t in the system such that y occurs, but not delayed, in t. A variable x *depends* on y iff it directly depends on it, or it depends on another variable which does.

An system of equations is *viciously circular* if at least one of its variable terms depends on itself.

For information to flow adequately through the circuit, every one of its circles must include at least one clock element. Otherwise, the circle becomes vicious. This means that (at least one unit of) time must pass before information closes a loop within the circuit.

Definitions 2.1.3

An equation x = t in a system is *determined* if every variable in t occurs at the left side of '=' in some equation of the system.

A system of equations is *well-formed* if it is not viciously circular and every one of its equations is determined.

3 Semantics

This language's semantics involves *streams of values*. Let A be a set of values. A stream of values defined over A is an ordered pair s = (*a*, *s'*) where *a* A, and *s'* is another stream over A. As its name suggests, a stream over A is an element of A followed by another stream. Streams have acquired special eminence in theoretical computer science because many programs have streams as their (ideal) output.

It is important to bring up some standard notions from stream theory that will be used in the following sections. First, for every set A, A* is the hyperset of streams s = (a, s'),

where a A, and s' A*. For example, 2* is the hyperset of streams like (1, (0, (1, (1, (1, (0, . . .), (0, (1, (0, (1, (0, (1, . . .), (1, (0, (1, (0, (1, (1, . . .), etc. Also, remember that 1^{st} and 2^{nd} are two important operations on streams. Operation 1^{st} takes the first element of a stream, while 2^{nd} takes it second, so that, for any stream *s* in A*, $1^{st}(s)$ gives an element of A and $2^{nd}(s)$ yields another stream.

For the purposes of this semantics, let D be the dominion of values that the abstract computer memory states are to take at a single time. The Interpretation function I will map every node of the language wffs into streams of those values, so that it traces the change through time of values in such memory state. This is the main advantage of this circular language.

Definition 3.1.1

Let V be a total function from **S** into D* and from **C** into D. It assigns streams of values of D to input symbols, and single values of D to initial state symbols. The rational behind this is that the computer may receive not a single value, but a stream of such as inputs. Also, since the final interpretation of the wff will also be a stream, it would allow for the composition of programs and wffs. Assigning single values to initial state constants is required to determine the initial state of the machine before running the program. To avoid unnecessary complications, a single complex function may play both roles.

Let L assign to every operational symbol O^n_i, a n-ary operation on D. Finally, for every operational symbol O^n_i let $/O^n_i/$ be the function from (D*) into D* such that $/O^n_i/(s_1, s_2, s_3, \ldots s_n) =_{def} (L(O^n_I\ 1^{st}(s_1),\ 1^{st}(s_2),\ 1^{st}(s_3), \ldots 1^{st}(s_n)),\ /O^n_i/(\ 2^{nd}(s_1),\ 2^{nd}(s_2),\ 2^{nd}(s_3), \ldots 2^{nd}(s_n)))$.

Definition 3.1.2

Let I_V be the interpretation function from terms into D* satisfying the following conditions:

1. If t **S**, then $I_V(t) =_{def} V(t)$.
2. If $t = \mathbf{O}^n_i(t_1, t_2, t_3, \ldots t_n)$, then $I_V(t) =_{def} /\mathbf{O}^n_i/(I_V(t_1), I_V(t_2), I_V(t_3), \ldots I_V(t_n))$.
3. If $t = [a, t_i]$, then $I_V(t) =_{def} (V(a), I_V(t_i))$.
4. For every equation $x = t$ in the system, $I_V(x) = I_V(t)$

When no ambiguity arises, drop the subscript V.

To evaluate the terms in a system of equations, it is recommendable to start from the input and initial state terms.

Lemma 3.1.3

I is defined for every term in a wff.

Proof: This is proved by a simple co-induction on the number of terms in the system of equations. The only cases of equations with one single term are of the form (1) $x = p$ or (2) $x = [a, x]$. It is straightforward to notice that, since V is a total function, I is defined for both

cases. The co-induction holds because, more complex terms are either of the form $\mathbf{O}^n_i(t_1, t_2, t_3, \ldots t_n)$ or $[a, t_i]$. Since no wff contains vicious circles, all bracketed terms are defined on their first element, which represents the information state of the computer at the beginning of the program. Hence, I(*a*) is defined for any term and variable in any equation of the well-formed system.

Definition 3.1.4

Let $I_V(x)$, where $x = t$ is the main equation of a wff, be the interpretation of that formula.

Proposition 3.2

Every wff has a unique stream over D as its interpretation. Every wff uniquely defines a stream over D.

Proof: Directly from lemma 3.1.3.

4 Logic

The expressive power of this logic is clear to see. As an example, in this section I offer a simple, circular proposicional logic over 2*, which is both sound and complete.

4.1 Syntax

Let O = { →, ¬, , } be the operator symbols of our language, where →, and are binary symbols and ¬ is unary. Let S, **X** and **C** be distinct sets of symbols, all of them different from **O**. For convenience, use Roman lower case letters *a*, *b*, *c*, . . . for initial state constants, *p*, *q*, *r*, . . . for input symbols, and *z*, *y*, *x*, . . . for term variables. Let Σ = **S** O **X** **C** {'=', '[', ']'} be the alphabet of our language.

Definitions 4.1.1

As usual, the operation letters applied to input and variable symbols generate the *terms*:

1. Variables and input letters are terms.
2. If t_1, t_2 are terms, then $t_1 \rightarrow t_2$, t_1 t_2 and t_1 t_2 are terms as well.
3. If t is a term, then ¬t is a term too.
4. If t is a term and *a* is an initial state constant, [*a*, t] is also a term.
5. An expression is a term only if it can be shown to be a term on the basis of conditions 1, 2, 3 and 4.

Equation, system of equations, main equation, vicious circle and well-formed formula are all defined according to definitions 2.1.1, 2.1.2 and 2.1.3 above.

Definition 4.1.2

Let $X_1 = \langle x_1 = t_1, E_1 \rangle$ and $X_2 = \langle x_2 = t_2, E_2 \rangle$ be two wffs of our language. The disjoint union of X_1 and X_2, written $X_1 + X_2$, is obtained by uniformly replacing every variable and clock in X_2 that occurs in X_1, by a new variable or clock, and then forming the set of its equations, including the main ones. The disjoint union of two wffs is not a wff itself. It is not even a system of equations, it is a mere set of equations. However, it will allow us to define operations on wffs parallel to those on terms.

Definition 4.1.3

Let $X_1 = \langle x_1 = t_1, E_1 \rangle$ and $X_2 = \langle x_2 = t_2, E_2 \rangle$ be two wffs of our language.

1. Let $X_1 \rightarrow X_2 =_{def} \langle x = x_1 \rightarrow x_2, E_1 + E_2 \}\rangle$,
2. Let $X_1 \ \ X_2 =_{def} \langle x = x_1 \ \ x_2, E_1 + E_2 \rangle$,
3. Let $X_1 \ \ X_2 =_{def} \langle x = x_1 \ \ x_2, E_1 + E_2 \rangle$,
4. Let $\neg X_1 =_{def} \langle x = \neg x_1, E_1 \ \ \{x_1 = t_1\}\rangle$.

Where x is always a new variable that does not occur in X_1, X_2 or its disjoint union.

Definition 4.1.4

Let $X_1 = \langle x_1 = t_1, E_1 \rangle$ be a wff of our language, and let *p* be an input symbol occurring in it. Define the looping of X_1 through *p* as the system $\langle x_1 = t_1, E_2 \quad \{x_2 = [a, t_2]\}\rangle$ where x and *a* are new symbols not occurring in X_1, E_2 results from E_1 and t_2 results from t_1 by replacing all occurrences of *p* in E_1 and t_1 for x_2.

Proposition 4.1.5

1. For every term t without variables $\langle x = t, \quad \rangle$ is a wff.
2. If X_1 is a wff and *a* is a clock symbol not occurring in X_1, $\langle x = [a, x_1], E_1 \ \ \{x_1 = t_1\}\rangle$ is a wff as well.
3. If X_1 and X_2 are wffs, $X_1 \ \ X_2$, $X_1 \ \ X_2$, $X_1 \rightarrow X_2$ and $\neg X_1$ are also wffs.
4. The looping of any wff is a wff, too.

Proof: Follows directly from our definition of wff.

Definition 4.1.6

Every wff resulting by repeated applications 1, 2, 3 and looping is called a *canonical* wff.

Theorem 4.1.7

For every wff X_1, there is a canonical wff X_2 (probably identical to X_1) such that, for all I and all V, $I_V(X_1) = I_V(X_2)$. In other words, every wff is semantically equivalent to a canonical wff.

Proof: It is easy to construct a canonical wff for every wff. First, eliminate all unnecessary equations form the system, that is, every equation $x_i = t_i$ such that the system's main variable

does not depend on x_i. The second step is to eliminate all circular equations by splitting them into two equations. Substitute every equation $x_i = t_i$ such that x_i occurs in t_i for the equation $x_i = t_j$, where t_j results from substituting every occurrence of x_i in t_i for x_j, and add $x_j = x_i$ to the system of equations. The resulting system of equations will be canonical and equivalent to the original.

4.2 Semantics

For the purposes of this semantics, let 2 = {0, 1} be the dominion of values that the abstract computer memory states are to take at a single time. The Interpretation function I will map every node of the language wffs into streams of ones and zeroes, so that it traces the change through time of values in such memory state.

Definition 4.2.1

Let V be a total function from **S** into 2* and from **C** into 2. It assigns streams of ones and zeroes to input symbols, and single values of one or zero to initial state symbols.

Let L assign to every logical connective the adequate n-ary logical operation on 2. In other words:

1. Let $L(\rightarrow, d_1, d_2) =_{def} 0$ if $d_1 = 1$ and $d_2 = 0$, and let $L(\rightarrow, d_1, d_2) = 1$ otherwise.
2. Let $L(\ , d_1, d_2) =_{def} 1$ if $d_1 = 1$ and $d_2 = 1$, and let $L(\ , d_1, d_2) = 0$ otherwise.
3. Let $L(\ , d_1, d_2) =_{def} 0$ if $d_1 = 0$ and $d_2 = 0$, and let $L(\ , d_1, d_2) = 1$ otherwise.
4. Let $L(\neg, d) =_{def} 0$ if $d = 1$ and Let $L(\neg, d) = 1$ otherwise.

Finally, for every operational symbol O^n_i let $/O^n_i/$ be the function from (D*) into D* such that:

1. $/\rightarrow/(d_1^*, d_2^*) =_{def} (L(\rightarrow, 1^{st}(d_1^*), 1^{st}(d_2^*), /\rightarrow/(2^{nd}(d_1^*), 2^{nd}(d_2^*))$
2. $/\ /(d_1^*, d_2^*) =_{def} (L(\ , 1^{st}(d_1^*), 1^{st}(d_2^*), /\ /(2^{nd}(d_1^*), 2^{nd}(d_2^*))$
3. $/\ /(d_1^*, d_2^*) =_{def} (L(\ , 1^{st}(d_1^*), 1^{st}(d_2^*), /\ /(2^{nd}(d_1^*), 2^{nd}(d_2^*))$
4. $/\neg/(d^*) =_{def} (L(\neg, 1^{st}(d^*)), /\neg/(2^{nd}(d^*))$

Definition 4.2.2

Let I_V be the interpretation function from terms into D* satisfying the following conditions:

5. If t **S**, then $I_V(t) =_{def} V(t)$.
6. If $t = (t_1 \rightarrow t_2)$, then $I_V(t) =_{def} /\rightarrow/(I_V(t_1), I_V(t_2))$.
7. If $t = (t_1 \ t_2)$, then $I_V(t) =_{def} /\wedge/(I_V(t_1), I_V(t_2))$.
8. If $t = (t_1 \ t_2)$, then $I_V(t) =_{def} /\vee/(I_V(t_1), I_V(t_2))$.
9. If $t = (\neg t_1)$, then $I_V(t) =_{def} /\neg/(I_V(t_1))$.
10. If $t = [a, t_i]$, then $I_V(t) =_{def} (V(a), I_V(t_i))$.
11. For every equation $x = t$ in the system, $I_V(x) =_{def} I_V(t)$

Finally, let $I_V(x)$, where $x = t$ is the main equation of a wff, be the interpretation of that formula. When no ambiguity arises, drop the subscript V.

4.3 Logical Implication

In order for this to be an actual logic, one must define a relation of logical implication. Following orthodoxy, we define logical implication as pointwise ≤.

Definition 4.3.1
For any two given streams d_1^* and d_2^* in 2^*, d_1^* logically implies d_2^*, written $d_1^* \leq d_2^*$, iff $1^{st}(d_1^*) \leq 1^{st}(d_2^*)$ and $2^{nd}(d_1^*) \leq 2^{nd}(d_2^*)$.

4.4 Proof Theory

Given the traditional nature of our semantics, any standard axiomatization of propositional logic may easily be adapted as an axiom system for this circular logic. Here, we borrow Elliot Mendelson's [7] system.

Definition 4.4.1
If X_1, X_2 and X_3 are wffs in our language, then the following are axioms of our axiomatic system:

1. $(X_1 \rightarrow (X_2 \rightarrow X_1))$
2. $((X_1 \rightarrow (X_2 \rightarrow X_3)) \rightarrow ((X_1 \rightarrow X_2) \rightarrow (X_1 \rightarrow X_3))$
3. $(((\neg X_1) \rightarrow (\neg X_2)) \rightarrow (((\neg X_2) \rightarrow X_1) \rightarrow X_2))$
4. $(X_1 \ \ X_2) \rightarrow (\neg (X_1 \rightarrow (\neg X_2)))$
5. $(\neg (X_1 \rightarrow (\neg X_2))) \rightarrow (X_1 \ \ X_2)$
6. $(X_1 \ \ X_2) \rightarrow ((\neg X_1) \rightarrow X_2)$
7. $((\neg X_1) \rightarrow X_2) \rightarrow (X_1 \ \ X_2)$

There are two rules of inference in our system. The first one is *modus ponens*: X_2 is direct consequence of X_1 and $X_1 \rightarrow X_2$. The second one, the rule *circular entailment*, is necessary to account for circular wffs: every looping of X_1 is a direct consequence of X_1.

Proposition 4.4.2
If Γ is a set of wffs, X_1 and X_2 are wffs, and X_2 is a logical consequence of the union of Γ and X_1, then $X_1 \rightarrow X_2$ is a logical consequence of Γ. The deduction theorem holds of this axiomatic system.
Proof: Since the system includes axioms schema 1 and 2, the deduction theorem may be proved following Herbrand proof from 1930.

Definition 4.4.3
Every wff of the language whose interpretation is the constant stream of ones $s^* = \langle 1, s^* \rangle$ is called a *tautology*.

Proposition 4.4.4
Every theorem of our system is a tautology.
Proof: Following the usual proof, it is easy to verify that every axiom has the constant stream of ones as its interpretation and that *modus ponens* preserves this semantic property. A little harder is to prove that the rule of circular entailment preserves it as well. The proof consists of noticing that, whatever the interpretation of the main variable in a wff, it is still one of the possible value assignments of an input symbol. The addition of the clock only avoids the vicious circularity, but it does not affect the formula's interpretation. Hence, if a wff is a tautology, any of its loopings is also a tautology. Therefore, every theorem of the system is a tautology.

Theorem 4.4.5
Every canonical tautology is a theorem of our system.
Proof: In the case of non-circular wffs, the result directly follows from the completeness of the aforementioned system for propositional calculus. I will waste no space here to go through such proof. The rest of the circular tautologies are obtained from the looping of non-circular tautologies. The inclusion of the rule of circular entailment allows for these tautologies to be obtained.

Proposition 4.4.6
A wff logically implies another if and only if the later is a logical consequence of the former.
Proof: From soundness (Proposition 4.4.3), completeness (Theorem 4.4.5), the inclusion of *modus ponens* and the theorem of deduction (Proposition 4.4.2).

5 Further Directions

I have shown that, using the traditional logical operators, it is possible to construct a sound and complete logic for streams on 2. I am currently working to produce a similar, more general soundness/completeness proof for any domain D.

Furthermore, it is also necessary to provide a well-defined mechanism for the implementation of this logic as a computational semantics, as it was originally designed to. It would be fruitful to find a programming language to serve as example to the advantages of using this logic for denotational semantics.

An even more relevant result would be to give a computational definition of the family of functions (over streams) expressible in this logic.

References

1. Abramsky, S. (ed.): Handbook of Logic in Computer Science. Clarendon Press, Oxford (1994)
2. Aczel, P.: Non-Well-Founded Sets. CSLI Lecture Notes **14**. CSLI Publications, Stanford (1988)
3. Barceló, A.: A Four Valued Circular Logic. Unpublished manuscript
4. Barwise, J. and Moss, L.: Hypersets. Mathematical Intelligencer **13** (1991) 31-41.
5. Barwise, J. and Moss, L.: Vicious Circles. On the Mathematics of Non-wellfounded Phenomena. CSLI Publications, Stanford (1996) 91-102
6. Dijkstra, E. W.: A Discipline of Programming. Prentice-Hall (1976)

Carlos Zozaya *et.al.* (ed.) *Memoria 3er Encuentro Internacional de Ciencias de la Computación* (Aguascalientes: SMCC/INEGI, 2001) 793-802