The Space-Time Semantics

José U. Ferreira Jr.

Temporary e-mail: *zeuxferreira@yahoo.com.br*

Abstract: *In order to formalize the semantics of mobile-agents technologies, this paper briefly introduces a logical language.*

1 An Informal Definition of the Space-Time Logic

The present section introduces a concise definition of the space-time classical logic.

Let **C** be the set of all formulae in the space-time logic, i.e. the language of the present logic. The syntax can be defined as follows:

Definition 1 *Let φ and ϕ be two formulae and α be a variable (a quantifier). Thus, the grammar for the space-time classical logic can be as follows:*

$$\begin{aligned}
&\varphi \longmapsto P \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi \mid \varphi \Leftrightarrow \varphi \mid (\exists\alpha)\ \varphi \mid (\forall\alpha)\ \varphi \\
&\varphi \longmapsto @s \cdot t[\varphi] \mid \phi \mid @s \cdot t[\ ] \\
&\phi \longmapsto @s \cdot t[\![\varphi]\!] \\
&\alpha \longmapsto x \mid y \mid \ldots
\end{aligned}$$

where φ is the starting symbol, P stands for a proposition or predicate, α denotes a quantified variable in the logic, $s \in \mathbb{R}^3$ or $s \subseteq \mathbb{R}^3$ and $t \in \mathbb{R}$ or $t \subseteq \mathbb{R}$, depending on the focus of attention, points or sets. Let ϕ stand for semantics, which can be seen as a function whose domain is in **C** and semantic image is a parameter of the present logic. More generally, having a set of operations $\mathbb{D}$, which includes space and time, i.e. $\mathbb{S} \times \mathbb{T}$, and a set of states of computation S, typically, the semantic domain is in $\mathbb{D} \times S$ whereas, given a semantic image $\mathbb{I}$, the space-time semantics of a language in the present logic is a function of type $\mathbb{D} \times S \longrightarrow \mathbb{I}$. Both the syntax and semantics are simple: if φ is a formula, then

$$@s \cdot t[\varphi]$$

is a formula in the present logic, where s indicates the place where φ holds, and t indicates the time when φ holds. A particular case is $@s \cdot t[\ ]$ which intuitively indicates that there is no assertion for space s and time t. This notation is capable of representing an empty data base or theory, for instance.

With $@s \cdot t[\![\varphi]\!]$, one is able to express "the meaning of φ at place s and time t".

Note that this language implicitly introduces a conjunction between the space and the time expressions, for every space-time formula. Now, for any formula or expression, everything happens intuitively in the same way as it would happen in the classical logics, except that now there are the variables of space and time, and that the classical logics formula or classical logics expression can be said to be valid *with respect to the new spatio-temporal context.*

2 A Space-Time Semantics

In this section, I illustrate with an application of the present logic to operational semantics of programming languages. Informally, I adopt the following conventions:

- σ: a state of the computation, a set, as usual.
- $\sigma(m/X)$: σ, in particular, $X = m \in \sigma$.
- $\langle \mathbf{true}, \sigma\rangle$: tt (the true value) in state σ.
- $\langle \mathbf{false}, \sigma\rangle$: ff (the false value) in state σ.
- $@s \cdot t[\![A]\!]$: the meaning of A at place s and time t.
- n, m: two real numbers.
- ϵ: time spent to execute the referred to operation.
- $\rightsquigarrow$: the evaluation of some operation (the left operand) in the program for obtaining its meaning (the right operand).

Traditionally, the formal semantics of programming languages do not require one to state the space-time components. For a semantic rule, it is assumed that the antecedents refer to executions before the execution of the statement that appears in the consequent in the rule. However, for mobile-code languages, it becomes important to make it explicit that such statements do not change the locality while some other statements do change locality. Further, places and time become important concepts on global environments such as the Internet.

Here, as an example, I present an operational semantics of the well known *while* language, extracted from [1] and others, with slight changes in addition to the present notation, to make it explicit that their constructs do not change locality.

2.1 The evaluation of Boolean expressions

$$\frac{@s \cdot t_0[\![\langle a_0, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![n]\!] \quad @s \cdot t_0 +_t \epsilon_0[\![\langle a_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![m]\!] \quad n = m}{@s \cdot t_0[\![\langle a_0 = a_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\mathbf{true}]\!]}$$

where ϵ (with any integer index) is the time for executing the operation. Notice that this notation allows the semantics to make explicit that a_0 is performed before a_1, whereas the form of evaluation could be different in another language. For a parallel version, for instance, the rule would be slightly different from the one above. The other semantic rule for $=$ is the following:

$$\frac{@s \cdot t_0[\![\langle a_0, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![n]\!] \quad @s \cdot t_0 +_t \epsilon_0[\![\langle a_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![m]\!] \quad n \neq m}{@s \cdot t_0[\![\langle a_0 = a_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\mathbf{false}]\!]}$$

For the less than or equal to operator, exist more two rules such as:

$$\frac{@s \cdot t_0[\![\langle a_0, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![n]\!] \quad @s \cdot t_0 +_t \epsilon_0[\![\langle a_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![m]\!] \\ n \leq m}{@s \cdot t_0[\![\langle a_0 \leq a_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\mathbf{true}]\!]}$$

$$\frac{@s \cdot t_0[\![\langle a_0, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![n]\!] \quad @s \cdot t_0 +_t \epsilon_0[\![\langle a_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![m]\!] \\ n > m}{@s \cdot t_0[\![\langle a_0 \leq a_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\mathbf{false}]\!]}$$

And two rules for the negation:

$$\frac{@s \cdot t_0[\![\langle b, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![\mathbf{true}]\!]}{@s \cdot t_0[\![\langle \neg b, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\mathbf{false}]\!]}$$

$$\frac{@s \cdot t_0[\![\langle b, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![\mathbf{false}]\!]}{@s \cdot t_0[\![\langle \neg b, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\mathbf{true}]\!]}$$

Conjunction:

$$\frac{@s \cdot t_0[\![\langle b_0, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![\alpha]\!] \\ @s \cdot t_0 +_t \epsilon_0[\![\langle b_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\beta]\!]}{@s \cdot t_0[\![\langle b_0 \textbf{ and } b_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1 +_t \epsilon_2[\![\alpha \ \wedge \ \beta]\!]}$$

2.2 The execution of commands

In this section, an operational semantics of the commands in the *while* language is presented.

Atomic commands:

$$@s \cdot t[\![\langle \mathbf{skip}, \sigma\rangle]\!] \rightsquigarrow @s \cdot t +_t \epsilon[\![\sigma]\!]$$

$$\frac{@s \cdot t_0[\![\langle a, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![m]\!]}{@s \cdot t_0[\![\langle X := a, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\sigma(m/X)]\!]}$$

Sequencing:

$$\frac{@s \cdot t_0[\![\langle c_0, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![\sigma'']\!] \\ @s \cdot t_0 +_t \epsilon_0[\![\langle c_1, \sigma''\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\sigma']\!]}{@s \cdot t_0[\![\langle c_0; c_1, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\sigma']\!]}$$

While-loops:

$$\frac{@s \cdot t_0[\![\langle b, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon[\![\mathbf{false}]\!]}{@s \cdot t_0[\![\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon +_t \Delta t[\![\sigma]\!]}$$

$$\frac{@s \cdot t_0[\![\langle b, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0[\![\mathbf{true}]\!] \\ @s \cdot t_0 +_t \epsilon_0[\![\langle c, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\sigma'']\!] \\ @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1[\![\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma''\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1 +_t \epsilon_2[\![\sigma']\!]}{@s \cdot t_0[\![\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle]\!] \rightsquigarrow @s \cdot t_0 +_t \epsilon_0 +_t \epsilon_1 +_t \epsilon_2[\![\sigma']\!]}$$

References

[1] Glynn Winskel. *The Formal Semantics of Programming Languages: an introduction.* The MIT Press, fourth edition, 1997.