A General Semantic Framework for Deterministic Concurrent Systems

CS 524
Lecture 17b
José Meseguer

We have just finished a review of two well-known models of parallel functional programming, namely, the rewrite theories $\mathcal{R}_{CL}$ and $\mathcal{R}_{\lambda}$ of combinatory logic and of the λ-Calculus. Part of this review has included the observation that the different variants of the λ-Calculus yield isomorphic categories of concurrent computations $\mathcal{T}_{\mathcal{R}_{\lambda}}^{CL}$, which we may safely restrict to closed terms to avoid useless technicalities.

But is that all? No, not at all. These are just some examples of a much bigger phenomenon, namely, deterministic systems. Such systems can be widely different. They can, for example be microprocessors, or sequential programming languages which are imperative in nature, or parallel algorithms for matrix operations or fast Fourier transforms. But what do we mean when we call all of these systems deterministic?

In all instances we mean something intuitively very simple:

Determinism 1: A system $\mathcal{S}$ is deterministic if from any state s_0, either there is a unique terminating state $s_0!$ such that $s \xrightarrow[\mathcal{S}]{*} s_0!$, or s cannot

reach any terminating state.

This notion, let us call it Determinism 1, captures quite well the idea of non-determinism; but it has a drawback. Which one? It does not fit well the case of never terminating systems. Why? Let us see an example.

(not)

Consider the set (semigroup) of non-empty strings $\{a,b,c\}^+$ where string concatenation — is associative and the rewrite theory on it with rewrite rules:

$a \to b \quad a \to c \quad b \to bb \quad c \to cc$

This system is clearly never terminating, that is, any string $w \in \{a,b,c\}^+$ can be rewritten by at least one of the above rules. Therefore, it satisfies the Deterministic 1 definition vacuously.

But it does not seem to fit well the intuitive notion of a deterministic system since there is a crucial moment when a clearly non-deterministic choice will decide two completely different evolutions of the system. Which choice? whether:

(1) to rewrite a to b by $a \to b$, or

(2) rewrite a to c by $a \to c$.

Since, clearly, in case (1) we have the infinite evolution:

$$a \to b \xrightarrow{*} b^n \to b^{n+1} \to \cdots$$

and in the other,

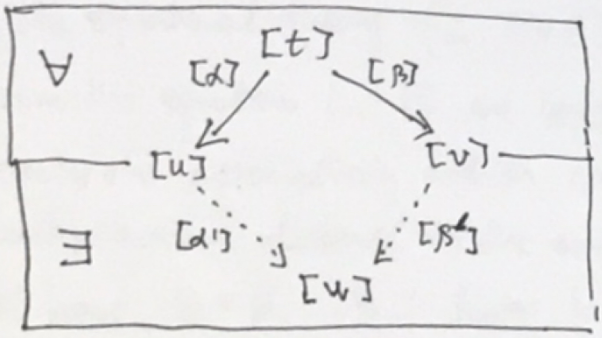
$$a \to c \xrightarrow{*} c^n \to c^{n+1} \to \cdots\cdots$$

So, calling this system "deterministic" is ludicrous.

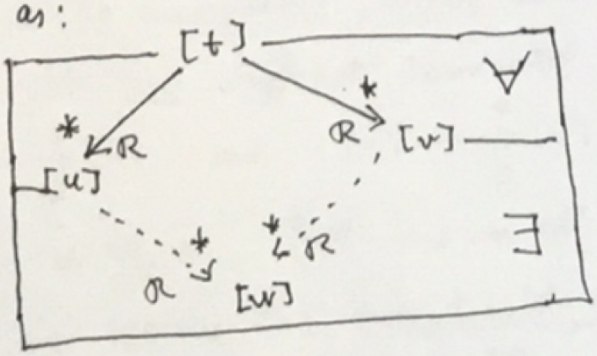
Back to the drawing board. So our Deterministic 1 notion will work just fine for terminating systems, but it is too loose for never-terminating ones or, more generally, for systems that sometimes terminate and sometimes do not. So, what do we do? Obviously to give the right definition! What is that? Obviously:

Determinism = Confluence. A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ is called confluent [technically, ground confluent] iff for any state $[t] \in T_{\Sigma/E}$ and any two concurrent computations $[t] \xrightarrow{[\alpha]} [u]$ and $[t] \xrightarrow{[\beta]} [v]$ in the category of computations $\mathcal{T}_{\mathcal{R}}$, there is

a state $[w] \in T_{\Sigma/E}$ and concurrent computations $[u] \xrightarrow{[\alpha']} [w]$ and $[v] \xrightarrow{[\beta']} [w]$. In picture:



Of course, since any concurrent computation in T_R, say $[u] \xrightarrow{[\gamma]} [u']$ has many possible interleavings, so that $[\gamma] = [\gamma_1; \cdots; \gamma_m]$, with $\gamma_i : [u_i] \to [u_{i+1}]$ a one-step rewrite with a single rule in R, we may, equivalently, describe the above picture in terms of sequences of one-step rewrites described by a one-step relation $\xrightarrow{}_R$ as:



The Equational Logic Connection

Consider a Maude functional module fmod $(\Sigma, E \cup B)$ endfm defined by the equational theory $(\Sigma, E \cup B)$. To keep life simple, assume the equations in E are unconditional. The set B, denotes the associativity and/or commutativity and/or unit/identity axioms declared in the module. Of course, we could have $B = \emptyset$. But how is this module executed in Maude? say, by the reduce command?

Obviously by executing the rewrite theory

$\vec{\mathcal{E}} = (\Sigma, B, \vec{E})$, where $\mathcal{E} = (\Sigma, E \cup B)$, and

where $\vec{E} = \{ u \to v \mid (u = v) \in E \}$.

But functional modules in Maude should not be defined in a state of drunkedness. Not any chaotic choice of equations E is acceptable, since then, very easily, we could get two different terminating computations

$[t] \xrightarrow[\vec{\mathcal{E}}]{*} [u]$ and $[t] \xrightarrow[\vec{\mathcal{E}}]{*} [v]$ with $[u] \neq [v]$,

i.e., with $u \neq_B v$. That is, in all Maude functional modules we require the executability condition [see Chapter 3 of Maude book] that $\vec{\mathcal{E}}$ is confluent!

The Church-Rosser Property

Q: What property does an equational theory $\mathcal{E} = (\Sigma, B \cup E)$ such that $\vec{\mathcal{E}} = (\Sigma, B, \vec{E})$ is confluent have?

A: It has the so-called Church-Rosser property that closely links equational deduction in $\mathcal{E}$ with rewriting in $\vec{\mathcal{E}}$:

Theorem (Church-Rosser Property). For an equational theory $\mathcal{E} = (\Sigma, B \cup E)$ such that $\vec{\mathcal{E}} = (\Sigma, B, \vec{E})$ is confluent we have the equivalence:

$$\forall\, t, t' \in T_\Sigma(X) \quad (t, t' \in T_\Sigma \text{ for ground confluence})$$

$$t =_{E \cup B} t' \iff \exists\, u \in T_\Sigma(X) \quad (u \in T_\Sigma \text{ for ground confl.})$$

$$\text{s.t. } [t] \xrightarrow[\vec{\mathcal{E}}]{*} [u] \xleftarrow[\vec{\mathcal{E}}]{*} [t']$$

Definition. Call $\vec{\mathcal{E}}$ terminating iff there are no infinite rewrite sequences $[t] \xrightarrow[\vec{\mathcal{E}}]{} [t_1] \xrightarrow[\vec{\mathcal{E}}]{} [t_2] \xrightarrow[\vec{\mathcal{E}}]{} [t_3] \cdots [t_n] \xrightarrow[\vec{\mathcal{E}}]{} [t_{n+1}] \cdots$

For confluent and terminating $\vec{\mathcal{E}}$ the above theorem gives us a decision procedure for $\mathcal{E}$-equality:

Theorem For $\mathcal{E} = (\Sigma, B \cup E)$ such that $\vec{\mathcal{E}} = (\Sigma, B, \vec{E})$ is confluent and terminating, then

$$t =_{E \cup B} t' \iff [t!_{\vec{\mathcal{E}}}] = [t'!_{\vec{\mathcal{E}}}],$$

where $t!_{\vec{\mathcal{E}}}$ is the unique (up to B-equality) terminating term [normal form] to which t can be rewritten.

That is, confluent equational theories $\mathcal{E} = (\Sigma, E \cup B)$, in other words, such equational theories such that $\vec{\mathcal{E}} = (\Sigma, B, \vec{E})$ is confluent, give us a much wider semantic framework to specify deterministic systems. Of course, both Combinatory Logic and the Lambda Calculus are special instances, since for

$\mathcal{R}_{CL} = (\Sigma_{CL}, \emptyset, R_{CL})$ we can specify the S, K, and I reduction rules as equations $E_{\mathcal{R}_{CL}}$ in a functional module, so that $R_{CL} = \vec{E}_{CL}$. In other words, we can either:

- Specify Combinatory Logic as a Maude functional module `fmod` $(\Sigma_{CL}, \vec{E}_{CL})$ `endfm`, or as a Maude system module `mod` $(\Sigma_{CL}, \emptyset, \vec{E}_{CL})$ `endm`. In the first case we compute with the `reduce` command, and in the second with the `rewrite` command, but the results are the same, since in both cases we use the rewrite theory $\vec{\mathcal{E}}_{CL} = (\Sigma_{CL}, \emptyset, \vec{E}_{CL})$.

The case of the λ-Calculus is slightly more interesting. To avoid any conditional equations that add unnecessary technicalities, let us consider the de Bruijn version. It is a rewrite theory

$$\mathcal{R}_{\lambda}^{DB} = (\Sigma_{\lambda DB}, Sub_{DB}, \{\beta\})$$

with Sub_{DB} the equations defining substitution, and a single rule, namely, β-reduction.

In this case, however, there are no axioms, i.e., $B = \emptyset$. Therefore, the equational theory is just

$$\mathcal{E}_{\lambda}^{DB} = (\Sigma_{\lambda DB}, Sub_{DB} \cup \{\beta_{EQ}\})$$

where β_{EQ} is β-reduction written as an equation. Then, $\overrightarrow{\mathcal{E}}_{\lambda}^{DB} = (\Sigma_{\lambda DB}, \emptyset, \overrightarrow{Sub}_{DB} \cup \{\beta\})$, which is slightly different from our original $\mathcal{R}_{\lambda}^{DB}$, since now we have explicitly added $\overrightarrow{Sub}_{DB}$ as rules. But there is no difference in execution: both Sub_{DB} and β are executed as rules either in mod $\mathcal{R}_{\lambda}^{DB}$ endm or in fmod $\mathcal{E}_{\lambda}^{DB}$ endfm, in one case with the rewrite command, whereas in the second case with the reduce command.

Summary: the Big Picture

1. A system specified as a rewrite theory $\mathcal{R}$ is <u>deterministic</u> iff [by definition] $\mathcal{R}$ is <u>confluent</u>.

2. Such systems can be specified either:

   (i) in rewrite logic itself by a <u>confluent</u> $\mathcal{R}$, or

   (ii) in equational logic by an equational theory $\mathcal{E} = (\Sigma, B \cup E)$ such that $\vec{\mathcal{E}} = (\Sigma, B, \vec{E})$ is <u>confluent</u>

3. Both Combinatory Logic and the Lambda Calculus <u>can</u> be specified either way in Maude: by method 2-(i) as we have done in previous lectures, or by method 2-(ii) as explained above.

4. The case of a rewrite theory $\mathcal{R}^{DB}_{\lambda} = (\Sigma_{\lambda DB}, Sub_{DB}, \{\beta\})$ where the equational part does not consists of axioms but of equations that are themselves confluent is quite interesting. More generally, theories $\mathcal{R} = (\Sigma, E \cup B, R)$ where $(\Sigma, B, \vec{E})$ is confluent. We will revisit this case when we study the notion of <u>coherence</u>.