Some Basic Properties of Concurrent Computations

CS 524
Lecture 22
J. Meseguer

Recall, from Lecture 11, that, given a rewrite theory, $\mathcal{R} = (\Sigma, E, R)$ [more generally, we could add to $\mathcal{R}$ a frozenness map $\phi : \Sigma \to \mathcal{P}(\mathbb{N})$ to restrict rewriting under some arguments, as described in a later lecture], a computation specification is a labeled arrow:

$$[u] \xrightarrow{\alpha} [v]$$

provable in the logic $\mathcal{L}(\mathcal{R})$ of $\mathcal{R}$ by the Reflexivity, Congruence, Replacement and Transitivity inference rules in fig. 6 of Lecture 11.

Notes: 1. Recall that, if a frozenness map $\phi : \Sigma \to \mathcal{P}(\mathbb{N})$ is given, the Congruence and Replacement inference rules are then restricted to forbid rewriting under frozen positions.

2. A somewhat subtle point worth keeping in mind is that, since for any $t' \in [t]_E$ in an E-equality equivalence class we have $[t] = [t']$, the Reflexivity rule could have been stated in the more explicit form: Reflexivity $\dfrac{}{[t] \xrightarrow{t'} [t]}$ where $t' \in [t] \in T_{\Sigma/E}$

that is, any $t' \in [t]$ can be used for an idle transition.

Finalley, recall that a <u>computation</u> of $\mathcal{R}$ has the form:

$$[u] \xrightarrow{[\alpha]} [v]$$

where $[\alpha]$ is an equivalence class of proof terms by the equivalences defined in Lecture 11, and computations then form a category $\mathcal{T}_{\mathcal{R}}$.

1. Interleaving Descriptions

Suppose $\mathcal{R}$ defines a Petri net with places a, b, c, d, e and transitions:

$[1] : ab \to bba$

$[2] : bc \to bcc$

$[3] : de \to dea$

Then we have the concurrent computation:

$$[abbcde] \xrightarrow{[[1][2][3]]} [aabbbde]$$

but we also have; for any permutation σ of the set $\{1, 2, 3\}$ computations

$$[abcde] \xrightarrow{[\sigma(1)]u_1 ; [\sigma(2)]u_2 ; [\sigma(3)]u_3} [aabbccde]$$

such that $[[1]\,[2]\,[3]] = [[\sigma(1)]u_1 ; \sigma[2]u_2 ; [\sigma(3)]u_3]$

For example, for $\sigma(1)=2,\ \sigma(2)=3,\ \sigma(3)=1$

we have:

$$[a\,b\,c\,d\,e] \xrightarrow{[2]\,abde} [a\,b\,c\,c\,d\,e] \xrightarrow{[3]\,abcc} [a\,a\,b\,c\,c\,d\,e] \to$$

$$\xrightarrow{[1]\,accde} [a\,a\,b\,b\,c\,c\,d\,e]$$

What all such descriptions, called <u>interleaving descriptions</u>, have in common is that the computation is described in a <u>sequentialized form</u>, so that a <u>single</u> <u>transition</u> takes place <u>at each step</u> in the sequential composition.

The obvious question to ask is: Is this always possible.

An the answer is: Yes!

<u>Sequentialization lemma</u>. Given a rewrite theory $\mathcal{R}=(\Sigma, E, R)$, any computation $[u] \xrightarrow{[\alpha]} [v]$ in $\mathcal{T}_{\mathcal{R}}$ is either:

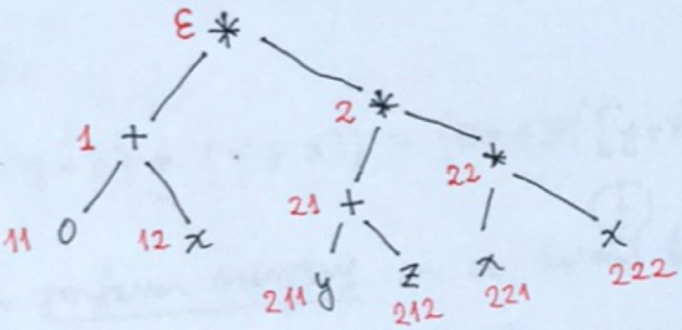
- $[\alpha]=[u]$ and $[u]=[v]$ (idle computation), or
- $[\alpha]=[\beta_1;\dots;\beta_n]$, $n \geqslant 1$, where each β is a <u>proof term</u> of the <u>form</u> $t[\ell(u_1,\dots,u_n)]_p$ involving variables $x_1,\dots,x_n$ with $(\ell : u \to v) \in R$ and $t, u_1,\dots,u_n \in T_\Sigma$, and p a <u>term position</u> in t, as explained in more detail below. Such a proof term is called a <u>one-step rewrite</u>.

Term positions and related notation

We can easily describe the <u>positions</u> of a term by displaying it as a <u>tree</u>, and numbering its nodes as follows:

For example, consider the term: $(0+x)*((y+z)*(x*x))$

Then its positions are:



That is, positions are strings in the monoid $\mathbb{N}^*$

Of course, we have a function

$$pos : T_\Sigma \longrightarrow \mathcal{P}(\mathbb{N}^*)$$
$$t \longmapsto pos(t)$$

assigning to each <u>term</u> t the <u>set</u> of <u>its positions</u>. For example:

$$pos((0+x)*((y+z)*(x*x))) = \{\varepsilon, 1, 2, 11, 12, 21, 211, 212, 22, 211, 212\}$$

Also, given a term t and a position $p \in pos(t)$ in it, then $t|_p$ denotes the <u>subterm</u> at that position. For example: $(0+x)*((y+z)*(x*x))\big|_{21} = y+z$

This notion of position allows us to decompose any term t such that $t|_p = u$ into a context and the subterm u "plugged" into such a context at position p. That is, t can be decomposed as: $t = t[u]_p$. when $u = t|_p$

For example,

$$(0+x)*((y+z)*(x*x)) = (0+x)*([y+z]_{21}*(x+x))$$

We can also perform surgery on a term t by replacing a term $u = t|_p$ by another term v at the same position. For example, we may replace in the above term the subterm $x+y$ at position 21 by the term $z*(0+x)$ to get:

$$(0+x)*([z*(0+x)]_{21}*(x+x)) = 0+x*((z*(0+x))*(x+x))$$

In general terms, if $t = t[u]_p$ and v is another term, we can replace u by v at p to get $t[v]_p$.

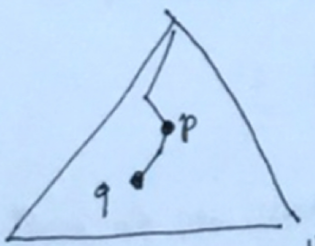
Given two positions $p, q \in Pos(t)$ of a term t, any of the following exclusive possibilities can happen:

1. p and q [not necessarily distinct] lay on the same path from the root of the term as a tree; we then call p and q <u>nested positions</u>, in the sense that either $p=q$, or the longer string, say q, is <u>nested</u> inside its prefix string p, i.e.

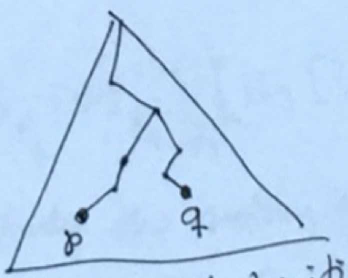
$$q = p\,q' \quad \text{for some } q',$$

maybe $q' = \varepsilon$, in case $p = q$

2. p and q are <u>on different paths</u> from the root. Then we call them <u>disjoint positions</u>.



nested positions



disjoint positions

Context notation can be generalized as follows:

1. If $p_1, \ldots, p_k$ are disjoint positions, we have a decomposition

$$t = t[u_1]_{p_1} \ldots [u_n]_{p_n} \quad \text{where } u_1 = t|_{p_1}, \ldots, u_n = t|_{p_n}$$

If $q = p\,p'$ with $p' \neq \varepsilon$ we also have a decomposition

$$t = t\,[\,u[v]_{p'}\,]_p$$

where $u = t|_p$ and $v = t|_{pp'}$

Of course, this can be iterated to $p_1 p_2 \cdots p_k$

with $t = t\,[\,u_1[u_2[\cdots[u_k]_{p_k}\cdots]_{p_2}]_{p_1}$

and $u_i = t|_{p_1 \cdots p_i}$

and we can "mix and match" parallel decompositions and nested ones, e.g:

$$t\,[\,u_1[\,u_2[v]_{p_3}[v']_{p_4}\,]_{p_1}[\,u_3[\,u_4[v'']\,]_{p_6}\,]_{p_5}\,]_{p_2}$$

Now we can understand more clearly the notion of a <u>one-step rewrite step</u> as a proof term of the form:

$$[\,t[u(v_1,..,v_n)]_p] \xrightarrow{t[\ell(v_1,..,v_n)]_p} [\,t[v(v_1,..,v_n)]_p\,]$$

where $\ell : u(x_1,..,x_n) \to v(x_1,..,x_n) \in R$

What this step does is to <u>replace</u> the term $u(v_1,..,v_n)$ at position p by the term $v(v_1,..,v_n)$ at same position

This notation has to be taken with a grain of salt, since, even if $x_1.., x_n$ appear in left-to right order in $u(x_1,..,x_n)$, the may not appear that way in $v(x_1..,x_n)$, since: (i) some variables may be dropped, and (ii) some variables may be repeated. What $u(v_1..,v_n)$ and $v(v_1..,v_n)$ really mean is: $u\{x_1 \mapsto v_1,.., x_n \mapsto v_n\}$, $v\{x_1 \mapsto v_1,.., x_n \mapsto v_n\}$, but the shorthand notation $u(v_1..,v_n)$ is of course simpler.

Sequential Systems

Sequential computation is a degenerate special case of concurrent computation. For example:

1. an automaton
2. a Turing machine [deterministic or non-deterministic]
3. a sequential programming language [with no parallelization optimizations such as speculative execution]

are all sequential systems.

But if rewriting logic is a semantic framework for concurrent computation, it should be possible to characterize the sequential system subcase.

How? Quite simply:

<u>Definition</u> A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ is called <u>sequential</u> if and only if:

1. It has <u>no sideways parallelism</u>, that is, no proof term of the form:

$$[u] \xrightarrow{t[l(u_1,.., u_n)]_p [l'(v_1,..,v_n)]_q} [v]$$

and p, q <u>disjoint</u>

with $l : u(x_1.., x_n) \to u'(x_1,.., x_n)$,

$l' : v(x_1,.., x_m) \to v'(x_1,.., x_m)$ rules in R

is possible at all.

2. It has no <u>nested concurrency</u>, i.e., no proof term of the form:

$$[u] \xrightarrow{t[l(u_1,.., u_i[l'(v_1,..,v_n)]_q,.., u_n)]_p} [v]$$

with p and $pp'q$ nested positions (for some p') is possible at all.

An obvious question is then the following:

If $\mathcal{R}$ is sequential, how do its computations in $\mathcal{T}_{\mathcal{R}}$ look like?

The answer can be given as follows:

<u>Theorem</u>. If $\mathcal{R} = (\Sigma, E, R)$ is sequential, then, any computation $[u] \xrightarrow{[\alpha]} [v]$ has a <u>unique</u> <u>interleaving</u> <u>description</u> as either

1. $[u] \xrightarrow{[u]} [v]$ idle computation, or
2. $$\begin{tikzcd} {[u]} \arrow[r, "{[\beta_1]}"] \arrow[rrr, bend right, "{[\alpha] = [\beta_1 ; \cdots ; \beta_n]}"'] & {[u_1] \cdots} & {[u_{n-1}]} \arrow[r, "{[\beta_n]}"] & {[v]} \end{tikzcd}$$ $n \geq 1$

where <u>unique</u> means <u>up to E-equality</u>, i.e.,

$[\beta_1 ; \ldots ; \beta_n] =_E (\beta'_1 ; \ldots ; \beta'_n)$

and of course <u>up to associativity of</u> $-;-$: $(\beta_1 ; \beta_2) ; \beta_3 = \beta_1 ; (\beta_2 ; \beta_3)$

<u>Proof Hints</u>

To prove this theorem, a useful notion is that of an <u>idle proof term</u>, which is a proof term built using the inference rules of $\mathcal{L}(\mathcal{R})$ using only the:

- <u>Reflexivity</u>
- <u>Congruence</u>, and
- <u>Transitivity</u>

rules, and showing that if $[u] \xrightarrow{\alpha} [v]$ with α an idle proof term, then $[u] = [v]$ and $[\alpha] = [u]$. Using this notion, the restrictions of a sequential $\mathcal{R}$, and induction [structural or on depth] ~~of~~ on proof terms, one can prove this theorem.