Sequential Process Expressions

January 7, 2002, 17:43

Uwe Nestmann

EPFL-LAMP

Repetition of Algebraic Notions (III)

congruence

- replacing equals for equals, i.e., preservation of equivalence under . . .

From Models to Languages

- represent system states by expressions
- hold information about structure and behavior
- “indicate” / infer the possible transitions

Sequential Process Expressions

$\mathcal{I}$	process identifiers	$A, B \ldots$
$\mathcal{N}$	names	$a, b, c \ldots$
$\overline{\mathcal{N}}$	co-names	$\overline{a}, \overline{b}, \overline{c} \ldots$
$\mathcal{L}$	labels (buttons)	$:= \mathcal{N} \cup \overline{\mathcal{N}}$
$\mathcal{A}$	actions	metavariables $\alpha, \beta \ldots \in \mathcal{L}$

- **finite sequences** $\vec{a}$ for *names* $a_1 \ldots, a_n$
- **parametric processes** $A\langle\, a, c\,\rangle$ with *name* parameters (*not* co-names, labels, $\ldots$)

Sequential Process Expressions (II)

Definition: The set $\mathcal{P}^{\mathrm{seq}}$ of seq. proc. exp. is defined (precisely) by the following BNF-syntax:

$$P \quad ::= \quad A\langle\,\vec{a}\,\rangle \quad \Big| \quad \sum_{i \in I} \alpha_i . P_i$$

where I is any finite indexing set. We use $P, Q, P_i \ldots$ to stand for process expressions.

$$\begin{array}{ccccc} I & = & \{1, 2\} & : & \ldots \\ I & = & \{\star\} & : & \ldots \\ I & = & \emptyset & : & \ldots \end{array}$$

Sequential Process Expressions (III)

- each process identifier A is assumed to have a **defining equation** (note the brackets)

$$A(\,\vec{a}\,) \stackrel{\mathrm{def}}{=} P_A$$

  where P_A is a summation, $\vec{a}$ includes $\mathrm{fn}(P_A)$.
- $\mathrm{fn}(P)$: the set of all of the **(free) names** of P
- $A\langle\,\vec{b}\,\rangle$ means the same as $\{^{\vec{b}}/_{\vec{a}}\}P_A$
- **substitution** $\{^{\vec{b}}/_{\vec{a}}\}P$ (for matching $\vec{b}$ and $\vec{a}$) replaces *all* occurrences of a_i in P by b_i.

Inductive Syntax

Is it well-defined ?

Definition:
The set $\mathrm{fn}(P)$ is defined inductively by:

$$\mathrm{fn}(A\langle\, \vec{a}\,\rangle) \quad \stackrel{\mathrm{def}}{=} \quad \ldots$$

$$\mathrm{fn}(\sum_{i\in I} \alpha_i.P_i) \quad \stackrel{\mathrm{def}}{=} \quad \ldots$$

$\ldots$

Inductive Syntax (II)

Define substitution formally, i.e., inductively !

$$\{{}^{b}\!/_{c}\}\alpha \stackrel{\text{def}}{=} \begin{cases} b & \text{if } \alpha = c \\ \overline{b} & \text{if } \alpha = \overline{c} \\ \alpha & \text{otherwise} \end{cases}$$

$$\{{}^{b}\!/_{c}\}A\langle\, \vec{a}\,\rangle \stackrel{\text{def}}{=} \ldots$$

$$\{{}^{b}\!/_{c}\}\sum_{i\in I} \alpha_i.P_i \stackrel{\text{def}}{=} \ldots$$

Inductive Syntax (III)

Define **simultaneous substitution** formally !

First, compute: $\{ {}^{b}/_{c}\,, {}^{a}/_{b}\, \} a.\overline{b}.c = \ldots$

$$\{{}^{\vec{b}}/_{\vec{c}}\}\alpha \stackrel{\text{def}}{=} \begin{cases} \ldots & \text{if } \alpha = c \\ \ldots & \text{if } \alpha = \overline{c} \\ \ldots & \text{otherwise} \end{cases}$$

$$\{{}^{\vec{b}}/_{\vec{c}}\}A\langle\, \vec{a}\, \rangle \stackrel{\text{def}}{=} \ldots$$

$$\{{}^{\vec{b}}/_{\vec{c}}\}\sum_{i\in I} \alpha_i.P_i \stackrel{\text{def}}{=} \ldots$$

Structural Congruence

Definition:

Two seq. proc. exp. P and Q
are **structurally congruent**, written $P \equiv Q$,
if we can transform one into the other
by replacing occurrences of $A\langle\,\vec{b}\,\rangle$ by $\{^{\vec{b}}/_{\vec{a}}\}P_A$,
or vice versa,
for arbitrary A defined by $A(\,\vec{a}\,) \stackrel{\text{def}}{=} P_A$.

Structural Congruence (II)

More “mathematically” (i.e., more precisely): the relation $\equiv$ is the smallest congruence generated$^{(*)}$ by the set of axioms

$$A\langle\,\vec{b}\,\rangle \;\equiv\; \{{}^{\vec{b}}/_{\vec{a}}\}P_A$$

induced from all A defined by $A(\,\vec{a}\,) \;\stackrel{\text{def}}{=}\; P_A$.

$(*)$: reflexive-symmetric-transitive context closure
("contexts" are expressions with single holes)

Structural Congruence (III)

$$\frac{}{P \equiv P} \qquad \frac{P \equiv Q}{Q \equiv P} \qquad \frac{P \equiv Q \qquad Q \equiv R}{P \equiv R}$$

$$\frac{P \equiv Q}{C[P] \equiv C[Q]}$$

where $C[\cdot]$ denote an arbitrary “process context” and $C[P]$ denotes filling the hole of $C[\cdot]$ with P.

Process Contexts

(* just as a hint on how to define them formally *)

Definition: A **process context** $C[\cdot]$ is (precisely) defined by the following syntax:

$$
\begin{array}{rcl}
C[\cdot] & ::= & [\cdot] \quad \Big| \quad \alpha.C[\cdot] + M \\
M & ::= & \sum_{i \in I} \alpha_i.P_i
\end{array}
$$

where I is any finite indexing set.

Note: summation is assumed to be commutative

Example

$$
\begin{aligned}
A(\,a, b\,) &\stackrel{\mathrm{def}}{=} a.A\langle\, a, b\,\rangle + b.B\langle\, a, a\,\rangle \\
B(\,c, d\,) &\stackrel{\mathrm{def}}{=} c.d.\mathbf{0}
\end{aligned}
$$

- exhibit some structural congruences
- rewrite $A\langle\, c, d\,\rangle$
  best without the use of process identifiers
- play with the variant

$$
A(\,a, b\,) \stackrel{\mathrm{def}}{=} a.A\langle\, b, a\,\rangle + b.B\langle\, a, a\,\rangle
$$

The LTS of Sequential Processes

Definition:
The LTS of sequential processes over $\mathcal{A}$ is defined to have **states** $\mathcal{P}^{\mathrm{seq}}$ and **transitions** as follows:
if $P \equiv \sum_{i \in I} \alpha_i.P_i$ then, for each $j \in I$, $P \xrightarrow{\alpha_j} P_j$.

Note: We distinguish the LTS of a *single process expression* (from the LTS of *all process expressions*) as just the part reachable from it.

Example: Boolean Buffer [Mil99, § 3.5]

$$\mathcal{N} := \{\ \mathsf{in}_i, \mathsf{out}_i \mid i \in \{0,1\}\ \}$$

$$s \in \{\epsilon, 0, 1, 00, 01, 10, 11\}$$

$$\mathsf{Buff}_s^{(2)} \stackrel{\text{def}}{=} \text{2-place buffer containing } s$$

$$\mathsf{Buff}^{(2)} \stackrel{\text{def}}{=} \sum_{i\in\{0,1\}} \mathsf{in}_i.\mathsf{Buff}_i^{(2)}$$

$$\mathsf{Buff}_i^{(2)} \stackrel{\text{def}}{=} \overline{\mathsf{out}_i}.\mathsf{Buff}^{(2)} + \sum_{j\in\{0,1\}} \mathsf{in}_j.\mathsf{Buff}_{ji}^{(2)}$$

$$\mathsf{Buff}_{ij}^{(2)} \stackrel{\text{def}}{=} \overline{\mathsf{out}_j}.\mathsf{Buff}_i^{(2)}$$

- modify $\mathsf{Buff}_s^{(2)}$ to release values in either order
- write an analogous definition for $\mathsf{Buff}_s^{(3)}$

Example: Scheduler (I) [Mil99, § 3.6]

- processes $P_i, 0 \leq i \leq n-1$ to be scheduled
- P_i starts by pressing a_i of the scheduler
- P_i completes by signalling b_i to the scheduler

- each P_i *must not* run two tasks at a time
- tasks of different P_i *may* run at a time

- a_i are required to occur cyclically (1 starts)
- for each i, a_i and b_i must occur cyclically
- permit maximal "pressure"

Example: Scheduler (II) [Mil99, § 3.6]

$i \in \{0 \ldots, n-1\} \quad X \subseteq \{0 \ldots, n-1\}$

$\mathsf{S}_{i,X} \stackrel{\text{def}}{=}$ scheduler, where i is next and X are running

---

$$\mathsf{S} \stackrel{\text{def}}{=} \mathsf{S}_{0,\emptyset}$$

$$\mathsf{S}_{i,X} \stackrel{\text{def}}{=} \begin{cases} \sum_{j \in X} b_j.\mathsf{S}_{i,X-j} & (i \in X) \\ \sum_{j \in X} b_j.\mathsf{S}_{i,X-j} + a_i.\mathsf{S}_{i+1 \bmod n, X \cup i} & (i \notin X) \end{cases}$$

- show that the scheduler is never deadlocked
- draw the transition graph for $n = 2$
- what is the difference when dropping $i \in X$?

Example: Counter [Mil99, § 3.7]

$$\begin{aligned} \mathsf{C} &\stackrel{\text{def}}{=} \mathsf{C}_0 \\ \mathsf{C}_0 &\stackrel{\text{def}}{=} \mathsf{inc}.\mathsf{C}_1 + \overline{\mathsf{zero}}.\mathsf{C}_0 \\ \mathsf{C}_{n+1} &\stackrel{\text{def}}{=} \mathsf{inc}.\mathsf{C}_{n+2} + \overline{\mathsf{dec}}.\mathsf{C}_n \end{aligned}$$

- generalize the counter to a stack of booleans
- modify the stack to become a queue