Process Algebra

Verification Using Model Checking - SS 2007

Stefan Blom

2007/06/25 14:13:35 v1.5

1 Basic Process Algebra

A process (p, py, pz) is a term built from several constructs.

The most basic constructs are

- *atomic actions* (a, b) taken from the set Act;
- *sequential composition* of two processes $(p \cdot q)$;
- *choice* between or *alternative composition* of two processes $(p + q)$.

In addition to these three construct, we also have the *deadlock* (δ).

Like in mathemathics, we often write $b \cdot c$ as bc.

To correspond to Uppaal, we consider a set of channels Chan and build our set of actions as

$$\mathsf{Act} = \{c!, c? \mid c \in \mathsf{Chan}\} \cup \{\tau, \mathsf{wait}\}$$

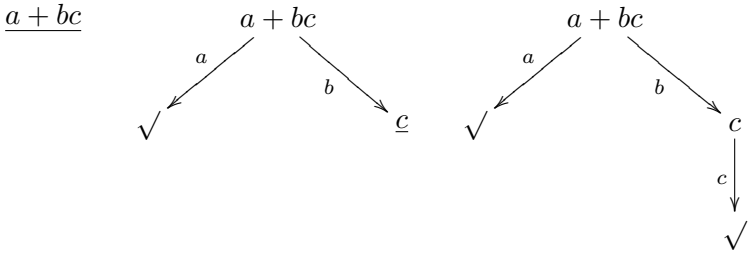
The two action τ and wait are special. The action τ is known as the invisible action. It is used to model the situation that we can observe that the system performs a step, but not which step. In some other process algebras (e.g. LOTOS) it is denoted i rather than τ. The waitaction denotes a time delay of one time unit.

The semantics of these processes is ternary relation $\rightarrow$. When we write $p \xrightarrow{a} q$, we mean that the process p can perform an a step to become the process q. We use an auxiliary symbol $\surd$ (pronounced tick) to denotes successful termination. When we write $p \xrightarrow{a} \surd$, we mean that p can terminate by performing an a step.

A process graph of a process p is a rooted directed graph built as follows:

1. The root is p and p is put on the todo list.
2. While the todo list is not empty, take an element q from the todo list. For every transition $q \xrightarrow{a} \surd$ add a new node $\surd$ and an edge, labeled with a from q to the new node. For every transition $q \xrightarrow{a} q'$, if no node q' exists create q' and put q' on the todo list. Regardless of the previous existence add an edge labeled a from q to q'

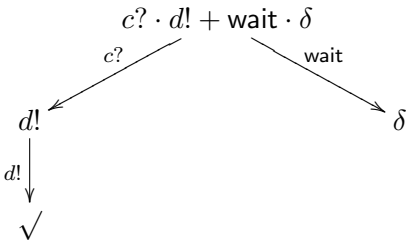
For example, if we denote the todo list by underlining then the process graph of $a + bc$ might be built as:



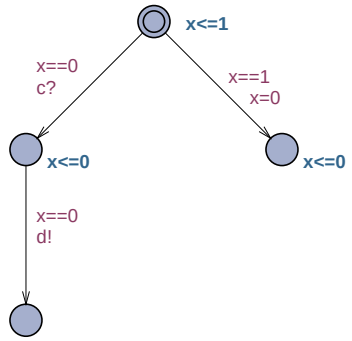
Process graphs can be translated to Uppaal templates as follows:

1. We have a clock `x`.
2. Every node in the graph becomes a location in the Uppaal template.
3. The root node becomes the initial location.
4. If an edge is labeled with wait, it gets a guard `x==1` and assigment `x=0` otherwise it gets a guard `x==1` (and no assignment).
5. If a node has a wait-edge it gets an invariant `x<=1` otherwise it gets an invariant `x<=0`.
6. The waitand τ labels are deleted.

Consider the process $c? \cdot d! + \mathsf{wait} \cdot \delta$. The process graph is



The Uppaal template corresponding to that graph is



A bisimulation is a relation R on processes, such that if $p \; R \; q$ then all of the following:

- If $p \xrightarrow{a} \surd$ then $q \xrightarrow{a} \surd$
- If $q \xrightarrow{a} \surd$ then $p \xrightarrow{a} \surd$
- If $p \xrightarrow{a} p'$ then $\exists q'.\ q \xrightarrow{a} q' \wedge p' \; R \; q'$
- If $q \xrightarrow{a} q'$ then $\exists p'.\ p \xrightarrow{a} p' \wedge p' \; R \; q'$

Two process p and q are bisimular ($p \leftrightarrows q$) if a bisimulation R exists, such that $p \; R \; q$.

2 Parallel Composition

The semantics of parallel composition is controlled by two functions the *communication function* $\gamma : \mathsf{Act} \times \mathsf{Act} \to \mathsf{Act} \cup \{\delta\}$ and the *interleaving function* $\gamma : \mathsf{Act} \to \mathsf{Act} \cup \{\delta\}$.

The communication function specifies if two actions can happen in parallel and if so, what the result is. We define this function as

$$\gamma(a,b) = \begin{cases} \tau & \text{, if } \exists c.\ \{a,b\} = \{c!, c?\} \\ \mathsf{wait} & \text{, if } \{a,b\} = \{\mathsf{wait}\} \\ \delta & \text{, otherwise} \end{cases}$$

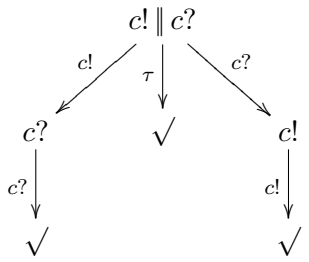
That is, a send and a receive can happen at the same time resulting in an invisible step; two wait steps can happen at the same time resulting in a wait step and no other actions can happen at the same time.

The interleaving function specifies if an action can happen by itself and if so what the result is. It is defined as

$$\gamma(a) = \begin{cases} \delta & \text{, if } a = \mathsf{wait} \\ a & \text{, otherwise} \end{cases}$$

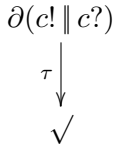
That is, all action can happen by themselves, except wait.

The process graph of $c! \parallel c?$ is



Given a set $H \subset \mathsf{Act} \setminus \{\tau\}$, we add the encapsulation of $\partial_H(p)$ to our syntax. Encapsulation is meant to insure that the actions in H can only be used internally. If we omit H then we mean $H = \{c!, c? \mid c \in \mathsf{Chan}\}$.

The process graph of $\partial(c! \parallel c?)$ is



Exercise 2.1
Use the axioms to prove the following processes equal to a process that uses actions, deadlock, sequential composition and choice only.

(a) $c!c! \parallel c?c?$

(b) $\partial(c!c! \parallel c?c?)$

(c) $\partial(c?c! \parallel c?c?)$

(d) $c!\delta \parallel c?c!$

(e) $\partial(\tau c! \parallel \tau c?c?)$

Exercise 2.2
Draw the process graphs of

(a) $c! \parallel d!(c? + d?)$

(b) $(c! + d?c!) \between d!c?$

(c) $\partial(c!d! \parallel (d! + c?d?))$

(d) $c! + d!c! \mid d?c?$

3 Recursive Definitions

We add process variables (X, Y, Z) as processes.

A *recursive definition* (E, F) is a set of equations whose left-hand sides are unique variables:

$$\{X_i = p_i \mid i \in I\} \text{ such that } X_i = X_j \Rightarrow i = j$$

A process is *guarded* if every recursion variable is preceeded by an action:

- a is guarded
- $p \cdot q$ is guarded if p is guarded
- $p \parallel q$ is guarded if p and q are guarded
- $p \between q$ is guarded if p is guarded
- $p \mid q$ is guarded if p and q are guarded
- $\rho_H(p)$ is guarded if p is guarded

A recursive definition is guarded if every right-hand side is guarded.

To reason about recursive definitions we have two principles. The *Recursive Declaration Principle*, which states that any equation can be used as an axiom. And the *Recursive Specification Principle*, which states that if a set of process terms over a recursive definition E satisfies a disjoint set of guarded equations F then the terms are equal to the corresponding variables in F.

Given a recursive definition E, some terms p_i over E and a guarded recursive definition $F = \{Y_i = q_i \mid i \in I\}$, such that none of the left-hand side variables of E occurs in F and none of the left-hand side variables of F occurs in E or some p_i. Then to prove that in $E \cup F$ we have $p_i = Y_i$, it suffices to prove that in E we can prove $F[Y_i = p_i]$. That is, in E we must prove $p_k = q_k[Y_i = q_i]$.

For example, given

$$E = \{X = a \cdot X\}$$

we can define $p_Y = p_Z = X$ and derive

$$p_Y = X = a \cdot X = a \cdot p_Z \text{ and } p_Z = X = a \cdot X = a \cdot p_Y$$

therefore we have

$$X = p_Y = Y, p_Z = Z$$

where

$$Y = a \cdot Z, Z = a \cdot Y$$

Another example. Given

$$X = a \cdot Y, Y = b \cdot a \cdot Y$$

we have

$$a \cdot Y = a \cdot b \cdot a \cdot Y$$

If we define $p_Z = a \cdot Y$ then this can be read as

$$p_Z = a \cdot b \cdot p_Z$$

so we have

$$p_Z = Z$$

where

$$Z = a \cdot b \cdot Z$$

Because $X = a \cdot Y = p_Z$, we have

$$X = Z$$

Direct proofs are not always possible. For example, to prove that

$$X = U$$

where

$$X = a \cdot X + b \cdot Y, Y = a \cdot Y + b \cdot X$$

and

$$U = a \cdot V + b \cdot U, V = a \cdot U + b \cdot V$$

we need to introduce

$$Z = a \cdot Z + b \cdot Z$$

and prove $X = Z$ and $Y = Z$ separately. The first case is

$$\underbrace{Z}_{p_X} = a \cdot \underbrace{Z}_{p_X} + b \cdot \underbrace{Z}_{p_Y} \text{ and } \underbrace{Z}_{p_Y} = a \cdot \underbrace{Z}_{p_Y} + b \cdot \underbrace{Z}_{p_X}$$

hence $p_X = X$ and $p_Y = Y$. The second case is similar.

Exercise 3.1
A singleton bag is bag which can contain at most one element. We define a singleton bag that can contain a 1 or a 2 as follows:

$$\mathrm{Bag}_1 = (c_1? \cdot c_1! + c_2? \cdot c_2!) \cdot \mathrm{Bag}_1$$

We can define a bag with unlimited capacity over $\{1, 2\}$ as

$$\mathrm{Bag}_\infty = c_1? \cdot (c_1! \parallel \mathrm{Bag}_\infty) + c_2? \cdot (c_2! \parallel \mathrm{Bag}_\infty)$$

(a) We can also define the singleton bag as:

$$\mathrm{Bag}' = c_1? \cdot c_1! \cdot \mathrm{Bag}' + c_2? \cdot c_2! \cdot \mathrm{Bag}'$$

Prove that the two definitions are equal. That is, prove that

$$\mathrm{Bag}_1 = \mathrm{Bag}'$$

(b) Prove that two infinite bags in parallel are equivalent to a single infinite bag. That is, prove that

$$\mathrm{Bag}_\infty = \mathrm{Bag}_\infty \parallel \mathrm{Bag}_\infty$$