1 The LPS-library

This document describes data types and algorithms of the LPS-library.

1.1 Terms

For an arbitrary term t we define $\mathcal{V}ar(t)$ as the set of data variables that occur in t. The result of substituting d' for d in a term t is denoted as $t[d := d']$. With $d \in Sub(t)$ we denote that data variable d is a subterm of data expression t.

1.2 Timed Linear processes

Let $\mathcal{D}$ be the set of all data expressions, with $\simeq$ an equivalence relation on $\mathcal{D}$. All data expressions have implicitly defined an associated sort. Let $\mathcal{V} \subset \mathcal{D}$ be the set of all data variables. Furthermore, let $\mathcal{A}$ be the set of action labels. A timed linear process X is a process equation of the following form:

$$X(d : D) = \sum_{i \in I} \sum_{e:E_i} c_i(d,e) \to a_i(f_i(d,e)) \lhd t_i(d,e) X(g_i(d,e)) + \sum_{i \in J} \sum_{e:E_i} c_i(d,e) \to \delta \lhd t_i(d,e), \tag{1}$$

and I and J are disjoint and finite index sets, and for $i \in I \cup J$:

- $c_i : D \times E_i \to \mathbb{B}$ is the condition function,
- $a_i(d,e)$ is a multi-action $a_i^1(f_i^1(d,e)) | \cdots | a_i^{n_i}(f_i^{n_i}(d,e))$, where $f_i : D \times E_i \to \mathcal{D}$ is the action parameter function,
- $t_i : D \times E_i \to \mathbb{R}$ is the time stamp function,
- $g_i : D \times E_i \to D$ is the next state function.

The components $d_i \in \mathcal{V}$ of vector $d = [d_1, \cdots, d_m]$ are called the *process parameters* of X.

1.3 Untimed Linear processes

Similarly, an untimed linear process X is a process equation of the following form:

$$X(d : D) = \sum_{i \in I} \sum_{e:E_i} c_i(d,e) \to a_i(f_i(d,e)) X(g_i(d,e)) + \sum_{i \in J} \sum_{e:E_i} c_i(d,e) \to \delta \quad , \tag{2}$$

with c_i, a_i and g_i defined as above.

1.3.1 State Space

An untimed linear process 2 with initial value $\overline{d} \in D$ defines a labeled transition system $M = (S, \Sigma, \to, s_0)$, where

- $S = D$ is the (possibly infinite) set of states
- $\Sigma = \{a_i(f_i(d,e)) | i \in I \wedge e : E_i\}$ is the (possibly infinite) set of labels
- $\to = \{(d, a_i(f_i(d,e)), g_i(d,e)) | i \in I \wedge e : E_i \wedge c_i(d,e)\}$
- $s_0 = \overline{d}$ is the initial state

For $R \subset S$ we define $next\_states(R)$ as $\{s \in S | \exists r \in R : r \to s\}$.

1.4 Global variables

In the mCRL2 tool set a linear process is parameterized with a finite set DC of so called *global* variables. This means that the expressions $c_i(d,e)$, $f_i(d,e)$, $g_i(d,e)$, and $t_i(d,e)$ may contain unbound variables from the set DC. Global variables have the implicit property that any two processes obtained by assigning values to them are strongly bisimulation equivalent. In many algorithms we will simply ignore global variables. In some cases, algorithms may assign values to some of the global variables.

1.5 Notations

Let p be an untimed linear process as defined in 2.

- A *constant parameter* of p is a parameter d_i that has a constant value for all reachable states of the corresponding state space, given an initial state $\overline{d} : D$.
- An *insignificant parameter* of p is a process parameter d_i such that for any two initial states that differ only at the value of d_i, the corresponding state spaces are isomorphic.
- Let p be a linear process, x a data variable and y a data expression. Then $p[x := y]$ is the linear process obtained from p by applying the substitution $x := y$ to all terms $c_i(d,e)$, $f_i(d,e)$, $g_i(d,e)$, and $t_i(d,e)$ that appear in the definition of p.

1.6 Linear processes

Linear process expressions in mCRL2 are expressions built according to the following syntax:

expression	C++ equivalent	ATerm grammar
$b(e)$	action(b,e)	Action
$\sum_v c \to a \lhd t \ P(g)$	summand(v, c, a, g)	LinearProcessSummand
$\sum_v c \to \delta \lhd t$	summand(v, c, δ)	LinearProcessSummand
$P(d := e)$	process_initializer$(f, d := e)$[1]	LinearProcessInit
$\sum_{i \in I} s_i$	linear_process(f, v, s)[2]	LinearProcess

where the types of the symbols are as follows:

a	a (timed) multi-action
b	a string (action name)
δ	a (timed) deadlock
P	a process identifier
e, f, g	a sequence of data expressions
d, v	a sequence of data variables
t	a data expression of type real
s	a sequence of summands
c	a data expression of type bool

A grammar for linear processes can be found in the Process implementation notes document.

1.7 Well typedness constraints

Not all linear processes that adhere to the grammar for linear processes are considered valid. A number of restrictions apply to make them valid input for the mCRL2 toolset. These restrictions are called *well typedness constraints*.

[1] Here f is a superset of the free variables appearing in e.

[2] Here $s = [s_1, \cdots, s_n]$ is a sequence of summands, v is the sequence of process parameters of the corresponding process, and f is a sequence of free variables appearing in s.

1.7.1 well typedness constraints for data specifications

- the domain and range sorts of constructors are declared in the data specification
- the domain and range sorts of mappings are declared in the data specification

1.7.2 well typedness constraints for a linear process

- the process parameters have unique names
- process parameters and summation variables have different names
- the left hand sides of the assignments of summands are contained in the process parameters
- the summands are well typed

1.7.3 well typedness constraints for linear process specifications

- the sorts occurring in the summation variables are declared in the data specification
- the sorts occurring in the process parameters are declared in the data specification
- the sorts occurring in the global variables are declared in the data specification
- the sorts occurring in the action labels are declared in the data specification
- the action labels occurring in the linear process are declared in the action specification
- the data specification is well typed
- the linear process is well typed
- the process initializer is well typed
- the global variables occurring in the linear process are declared in the global variable specification
- the global variables occurring in the initial process are declared in the global variable specification
- the global variables have unique names

1.7.4 well typedness constraints for other types

- the sorts of the left and right hand side of an assignment are the same
- the time of a summand has type Real
- the condition of a summand has type Bool
- the set of left hand sides of assignments in an action summand or process initializer does not contain duplicates

2 Algorithms

We now define two operations on linear processes: removing (insignificant) parameters and removing constant parameters. Let $X(d : D)$ be an linear process as defined in 2, or 1 and let $\{d_{j_1}, \cdots, d_{j_m}\}$ be a set of insignificant parameters of X. Then we define RemoveParameters$(p, \{d_{j_1}, \cdots, d_{j_m}\})$ as a linear process obtained from $X(d : D)$ by removing $\{d_{j_1}, \cdots, d_{j_m}\}$ from the process parameters of X, and by replacing each term $c_i(d, e)$, $f_i(d, e)$, $g_i(d, e)$, or $t_i(d, e)$ that appears in the definition of X, and that has one of the variables $d_{j_1}, \cdots, d_{j_m}$ as a subterm by a term $c'_i(d, e)$, $f'_i(d, e)$, $g'_i(d, e)$, or $t'_i(d, e)$ that does not have one of the variables $d_{j_1}, \cdots, d_{j_m}$ as a subterm, and such that the remaining process is strongly bisimulation equivalent to $X(d : D)$[3].

Let $X(d : D)$ be an linear process as defined in 2, or 1 and let $\{d_{j_1}, \cdots, d_{j_m}\}$ be a set of constant parameters of X, given the state $\overline{d} : D$. Then we define RemoveConstantParameters$(p, \{d_{j_1}, \cdots, d_{j_m}\}, \overline{d})$ as a linear process obtained from $X(d : D)$ by removing $\{d_{j_1}, \cdots, d_{j_m}\}$ from the process parameters of X, and by replacing each term $c_i(d, e)$, $f_i(d, e)$, $g_i(d, e)$, or $t_i(d, e)$ that appears in the definition of X by a term $c'_i(d, e)$, $f'_i(d, e)$, $g'_i(d, e)$, or $t'_i(d, e)$ that does not have one of the variables $d_{j_1}, \cdots, d_{j_m}$ as a subterm, and such that the remaining process is strongly bisimulation equivalent to $X(d : D)$[4].

2.1 Parelm

Let P be the stochastic linear process

$$P(d : D) = \sum_{i \in I} \sum_{e : E_i} c_i(d, e_i) \to a_i(f_i(d, e_i)) \triangleleft t_i(d, e_i) . \frac{p_i(d, e_i, h_i)}{h_i : F_i} P(g_i(d, e_i, h_i)),$$

with initial state

$$P_{init} = \frac{p(h)}{h : F} P(g(h)),$$

with $d = [d_1, \ldots, d_n]$ the process parameters of P. The algorithm Parelm is used to find insignificant process parameters that can be eliminated from P without altering the corresponding state space. It is not guaranteed that all insignificant process parameters are detected.

Parelm implementations We define the following implementations of parelm:

Parelm1(P, P_{init})
$G := \{k \in [1, \ldots, n] \mid \exists_{i \in I} : d_k \in \mathcal{FV}(c_i(d, e_i)) \cup \mathcal{FV}(f_i(d, e_i)) \cup \mathcal{FV}(t_i(d, e_i)) \cup \mathcal{FV}(p_i(d, e_i, h_i))\}$
do
  $\Delta G := \{k \in [1, \ldots, n] \mid \exists_{l \in G}\ \exists_{i \in I} : d_k \in \mathcal{FV}(g_i(d, e_i, h_i)_l)\}$
  $G := G \cup \Delta G$
while $(\Delta G \neq \emptyset)$
return $\{d_{j_1}, \cdots, d_{j_m}\}$,

where $\{j_1, \cdots, j_m\} = G$. Note that $g_i(d, e)_l$ is the l-th component of the vector of terms $g_i(d, e)$.

Parelm2(P, P_{init})
$W := \{d_k \mid \exists_{i \in I} : d_k \in \mathcal{FV}(c_i(d, e_i)) \cup \mathcal{FV}(f_i(d, e_i)) \cup \mathcal{FV}(t_i(d, e_i)) \cup \mathcal{FV}(p_i(d, e_i, h_i))\}$
$V := \{d_1, \ldots, d_n\}$
$E := \{(d_j, d_k) \mid \exists_{i \in I} : d_j \in \mathcal{FV}(g_i(d, e_i, h_i)_k\}$
$R := \{d_k \mid$ graph $G = (V, E)$ contains a directed path $w, \cdots, d_k$ for some $w \in W\}$
return R

[3] A more formal definition of this is welcome.
[4] Or should this be restricted to the result of substituting all the constant values, and possibly applying the rewriter to it?

2.2 Constelm

Let P be the stochastic linear process

$$P(d:D) = \sum_{i \in I} \sum_{e:E_i} c_i(d,e_i) \rightarrow a_i(f_i(d,e_i)) \lessdot t_i(d,e_i).\frac{p_i(d,e_i,h_i)}{h_i:F_i} P(g_i(d,e_i,h_i)),$$

with initial state

$$P_{init} = \frac{p(h)}{h:F} P(g(h)),$$

with $d = [d_1, \ldots, d_n]$ the process parameters of P. The algorithm CONSTELM is used to find process parameters that have a constant value. If needed the global variables of P will be assigned constant values. The result is a substitution σ that assigns these constant values to the corresponding process parameters and global variables. It is not guaranteed that all constant parameters are detected.

Constelm implementations Then we define the following implementation of CONSTELM:

```
CONSTELM(P, P_init, V, R)
r := R(g(h))
G := {d_1, ..., d_n}
σ := ε
for j := 1 to n do
    σ[d_j] := r_j
    undo[j] := ∅
do
    ΔG := ∅
    for i ∈ I do
        if R(σ(c_i(d, e_i))) ≠ false
            for d_j ∈ G \ ΔG do
                let z = R(g_i(d, e_i, h_i)_j, σ)
                if z ≠ σ(d_j)
                    if z ∈ V
                        σ[z] := r_j
                        undo[j] := undo[j] ∪ {z}
                    else
                        ΔG := ΔG ∪ {d_j}
                        σ[d_j] := d_j
                        for w ∈ undo[j] do
                            σ[w] := w
                        undo[j] := ∅
    G := G \ ΔG
while (ΔG ≠ ∅)
return σ
```

where V is the set of global variables of the linear process P, where R is a rewriter and ϵ is the empty substitution (i.e. the identity function). The notation $\sigma[v] := w$ means $\sigma = \sigma'$ with

$$\begin{cases} \sigma'(x) = \sigma'(x) \text{ if } x \neq v \\ \sigma'(x) = w \text{ if } x \neq v \end{cases}$$

An alternative implementation may look like this:

```
CONSTELM2(p, d̄)
R := {d}‾
G := {1, ..., n}
do
    ΔR := next_states(p, R)\R
    if ∃_{i∈I}.∃_{j∈G}.∃_{r∈ΔR}.r_j ≠ d̄_j
        G := G\{j}
    fi
    R := R ∪ ΔR
while (ΔR ≠ ∅)
return REMOVECONSTANTPARAMETERS(p, {d_{j_1}, ··· , d_{j_m}}, d̄),
```

where $\{j_1, \cdots, j_m\} = G$.

2.3 Conversion to linear process

The function lin converts a process expression to linear process format, if it is linear.

$$
\begin{aligned}
lin(\text{if\_then}(c, \delta)) &= \text{summand}([], c, \delta) \\
lin(\text{if\_then}(c, \text{seq}(x, P(g))) &= \text{summand}([], c, lin(x), g) \\
\text{add\_summand\_variables}(\text{summand}(v, c, \overrightarrow{d}, g), w) &= \text{summand}(v + w, c, \overrightarrow{d}, g) \\
\text{add\_summand\_variables}(\text{summand}(v, c, \delta), w) &= \text{summand}(v + w, c, \delta) \\
lin(\text{sum}(v, x) &= \text{add\_summand\_variables}(lin(x), w) \\
lin(\text{seq}(x, P)) &= lin(x) \\
\text{set\_multi\_action\_time}(\text{multi\_action}([a_1, \cdots, a_n]), t) &= \text{multi\_action}([a_1, \cdots, a_n], t) \\
lin(\text{at\_time}(x, t)) &= \text{set\_multi\_action\_time}(lin(x), t) \\
\text{multi\_action}([a_1, \cdots, a_m]) + \text{multi\_action}([a_{m+1}, \cdots, a_n]) &= \text{multi\_action}([a_1, \cdots, a_n]) \\
lin(\text{sync}(x, y)) &= lin(x) + lin(y) \\
lin(a) &= \text{multi\_action}([a]) \\
lin(\tau) &= \text{multi\_action}([]) \\
lin(\text{choice}(\text{choice}(x, y), z)) &= lin(\text{choice}(x, y)) + [lin(z)] \\
lin(\text{choice}(x, \text{choice}(y, z))) &= [lin(x)] + lin(\text{choice}(y, z)) \\
lin(\text{choice}(x, y)) &= [lin(x), lin(y)]
\end{aligned}
$$