lpsconstelm

F.P.M. Stappers

January 19, 2019

Abstract

This documentation describes the implementation and test cases of the tool *lpsconstelm* in the mCRL2 toolset. Basically, *lpsconstelm* is a tool which eliminates constant parameters in a linear process specification (LPS).

Contents

1 Introduction

The *lpsconstelm* tool is a tool for mCRL2 studio. The tool is a filter which reads from file *input.lps*, which is a file in *.lps* format [3]. We make use of the *LPS framework* [2] to read *input.lps*. For each detected constant process parameter the filter substitutes its constant value and removes the process parameter from the linear process specification. After the algorithm (Section 7.1) terminates, *lpsconstelm* will write the output to an output file *output.lps* (in the *.lps* format.)

2 Definitions

The equation below is a linear process specification in mCRL2:

Definition 2.1. linear process specification (LPS)

$$X(\overrightarrow{d:D}) = \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} c_i(\overrightarrow{d,e_i}) \to (a_i^1(\overrightarrow{f_{i,1}}(\overrightarrow{d,e_i}))|\ldots|a_i^{n(i)}(\overrightarrow{f_{i,n(i)}}(\overrightarrow{d,e_i})))^\leftarrow\ t_i(\overrightarrow{d,e_i}) \cdot X(\overrightarrow{g_i}(\overrightarrow{d,e_i}))+$$
$$\sum_{j\in J} \sum_{\overrightarrow{e_j:E_j}} c_j(\overrightarrow{d,e_j}) \to (a_j^1(\overrightarrow{f_{j,1}}(\overrightarrow{d,e_j}))|\ldots|a_j^{n(j)}(\overrightarrow{f_{j,n(j)}}(\overrightarrow{d,e_j})))^\leftarrow\ t_j(\overrightarrow{d,e_j})+$$
$$\sum_{\overrightarrow{e_\delta:E_\delta}} \overrightarrow{c_\delta}(\overrightarrow{d,e_\delta}) \to \delta^\leftarrow\ t_\delta(\overrightarrow{d,e_\delta})$$

Where I and J are disjoint.

For the description of the *lpsconstelm* tool we only use those elements of an LPS which are relevant. So we use a simplified[1] representation of an LPS .

Definition 2.2 (Simplified LPS)**.**

$$X(\overrightarrow{d:D}) = \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} c_i(\overrightarrow{d,e_i}) \to X(\overrightarrow{g_i}(\overrightarrow{d,e_i}))$$

If we talk about an LPS in this article we refer to Definition 2.2. The different states of a process are represented by the data vector parameter $\overrightarrow{d:D}$. Type $\overrightarrow{D}$ may be a Cartesian product of n data types, meaning that $\overrightarrow{d}$ is of a tuple $(d_1,\ldots,d_n)$. Let $x \in \{1,\ldots,|\overrightarrow{d}|\}$. The x^{th} element in the tuple is notated: $\overrightarrow{d}_{.x}$. The LPS expresses that in state $\overrightarrow{d}$ it can reach the new state $\overrightarrow{g_i}(\overrightarrow{d,e_i})$ under the condition that $c_i(\overrightarrow{d,e_i})$ is *true*. So, for each summand i from I we have a function $\overrightarrow{g_i} : \overrightarrow{D} \times \overrightarrow{E_i} \to \overrightarrow{D}$ and a function $c_i : \overrightarrow{D} \times \overrightarrow{E_i} \to \mathbb{B}$. If we want to speak about a specific statespace, which is generated from an LPS, we can instantiate this state space with the initial state $X(\overrightarrow{v_0})$.
For an more detailed explanation of linear process specifications we refer to [1].

3 Free Variables

3.1 Explanation

If a parameter at a certain state does not influence the behavior of an LPS at that specific state, we can assign to this process parameter a *free variable*. With these *free variables* we are able to

[1] Multiactions, time, deadlock and termination are not taken into account

model that process parameter can have values, which will not influence the behavior of an LPS.

3.2 Assumptions

We make certain assumptions about *free variables* which occur in an LPS, otherwise reasoning about LPS's would be to complicated. The first assumption is that the value of these variables will not influence the behavior of the LPS. Another assumption is that *free variables* only occur once in an LPS. Therefore each *free variable* is unique. Also we assume that each value of a state is either a closed term or a free variable.

For further reading about linear process specification and free variables please refer to [4]

4 lpsconstelm definition

4.1 lpsconstelm definition - without free variables

In this definition we do not consider free variables. A parameter of an LPS can be replaced by its initial value if it can be determined that this parameter remains constant throughout the run of any trace, starting at the initial state. The elimination of constant parameters does not reduce the resulting state space, however it may reduce the time and space needed to for example generate a state space from an LPS.
If we have infinite time and space we can inspect each state and check whether a process parameter changes throughout the execution of the process. For this we define the reachable state set R for each state ($\overrightarrow{v_0}$).

Definition 4.1. Let L be an LPS, where:

- I is the set of summand indices,
  $\overrightarrow{E_i}$ is the set of sum variables of summand i,
  c_i is the condition function of summand i and
  $\overrightarrow{g_i}$ is the next state function of summand i.

$R_L(\overrightarrow{v_0})$ is the smallest set for which the following holds:

- $\overrightarrow{v_0} \in R_L(\overrightarrow{v_0})$
- For all $i \in I$ and $\overrightarrow{e_i} \in \overrightarrow{E_i}$, if $\overrightarrow{v} \in R_L(\overrightarrow{v_0})$ and $c_i(\overrightarrow{v, e_i})$ holds, then $\overrightarrow{g_i}(\overrightarrow{v, e_i}) \in R_L(\overrightarrow{v_0})$

Definition 4.2. We define $S_L(\overrightarrow{v_0})$ to be the set of indices of process parameters for LPS L which are constant. That is:

$$S_L(\overrightarrow{v_0}) = \{j \in \{1, \ldots, |\overrightarrow{v_0}|\} | (\forall_{\overrightarrow{v} \in R_L(\overrightarrow{v_0})} (\overrightarrow{v}_{.j} = \overrightarrow{v_0}_{.j}))\}$$

To get $S_L(\overrightarrow{v_0})$ we have to compute $R_L(\overrightarrow{v_0})$. This means we have to generate the entire state space starting at state $\overrightarrow{v_0}$. Generating the entire state space can take a lot of time and space; it might take infinite time and/or space. However if we can make an approximation of Definition 4.2 we do not necessarily have to inspect all states. Note that if we use an approximation there are cases in which not all constant parameters can be found. In practice we do not mind, because these are hard to find and the cost/benefit is to small.

4.2 lpsconstelm definition - with free variables

A process parameter of an LPS can be replaced by its constant value if it can be determined that this parameter remains constant throughout any run of the process, starting at the initial state. However if a process parameter is initially a free variable and the process parameter remains constant, the specific assigned value or the initial *free variable* value is used.

We define the reachable state set R^{FV} for each state $(\overrightarrow{v_0})$.

Definition 4.3. Let L be an LPS, where

- I is the set of summand indices,
  $\overrightarrow{E_i}$ is the set of sum variables of summand i,
  c_i is the condition function of summand i and
  $\overrightarrow{g_i}$ is the next state function of summand i.
  FV is the set of *free variables* occurring in L.

$R_L^{FV}(\overrightarrow{v_0})$ is the smallest set for which the following holds:

- $\overrightarrow{v_0} \in R_L^{FV}(\overrightarrow{v_0})$
- For all $i \in I$ and $\overrightarrow{e_i} \in \overrightarrow{E_i}$, if $\overrightarrow{v} \in R_L^{FV}(\overrightarrow{v_0})$ and $c_i(\overrightarrow{v, e_i})$ holds, then $\overrightarrow{g_i}(\overrightarrow{v, e_i}) \in R_L^{FV}(\overrightarrow{v_0})$

Definition 4.4. We define $S_L^{FV}(\overrightarrow{v_0})$ to be the set of indices of process parameters which are constant. That is:

$$S_L^{FV}(\overrightarrow{v_0}) = \{j \in \{1, \ldots, n\} | \forall_{\overrightarrow{v}, \overrightarrow{w} \in R_L^{FV}(\overrightarrow{v_0})}(\overrightarrow{v}_{.j} \notin FV \wedge \overrightarrow{w}_{.j} \notin FV \rightarrow (\overrightarrow{v}_{.j} = \overrightarrow{w}_{.j})))\}$$

To get $S_L^{FV}(\overrightarrow{v_0})$ we have to compute $R_L^{FV}(\overrightarrow{v_0})$, and this means we have to generate the entire state space starting from state $\overrightarrow{v_0}$. However we interested in an approximation. We take proposal 3 (Section 5.7) as a starting point for constructing Proposal 4 (Section 5.12).

5 Proposals

In this section we speak about different techniques to find an algorithm that find as many constant process parameters in polynomial time. These proposals are all estimations, so we can reduce complexity of the given definitions in Section 4. Each proposal contains a small description, which describes how a certain proposal was born.

5.1 Proposal 1 - without free variables

Because variables can occur in non-normal forms we introduce a form function which allows us to take the normal form of such a data term. We write $NF(t)$ for the normal form of data term t. Let $j \in \{1, \ldots, n\}$ and let $\overrightarrow{d}$ be state vector. We assume $\overrightarrow{d}_{.j}$ is a normal form. If $\overrightarrow{d}_{.j}$ is equal to $NF(\overrightarrow{g_i}(\overrightarrow{d, e_i})_{.j})$ for all $i \in I$ we know argument at position j is constant. Each summand has a condition c_i. If a condition $c_i(\overrightarrow{d, e_i})$ does not hold, the next state $X(\overrightarrow{g_i}(\overrightarrow{d, e_i}))$ can not be reached. This is why we omit comparing each process parameter in $\overrightarrow{d}$ with such a $X(\overrightarrow{g_i}(\overrightarrow{d, e_i}))$.

However some conditions $NF(c_i(\overrightarrow{d, e_i}))$ might never be *true*, so it useless to compare $NF(\overrightarrow{d}_{.j})$ with $NF(\overrightarrow{g_i}(\overrightarrow{d, e_i})_{.j})$ if $NF(c_i(\overrightarrow{d, e_i}))$ is not "*true*".

Definition 5.1. Let L be an LPS and we define S^1_L:

$$S^1_L = \{j \in \{1, \ldots, n\} | \forall_{i \in I}(\forall_{\overrightarrow{d}:\overrightarrow{D}}((NF(c_i(\overrightarrow{d, e_i})) = \text{"}true\text{"}) \Rightarrow (NF(\overrightarrow{d}_{.j}) = NF(\overrightarrow{g_i}(\overrightarrow{d, e_i})_{.j}))))\}$$

If we take a look at Example 5.2, we see it might be possible to improve our proposal to detect more constant process parameters.

Example 5.2. Let $CE1$ be

```
proc P(x : Nat) = true -> P(x := 2 * x);
init P(x : = 0);
```

If we use Definition 5.1, we get:

$$\begin{aligned} & S^1_{CE1} \\ \equiv & \\ & \{j \in \{1\} | \forall_{x:Nat}(x = 2 * x)\} \\ \equiv & \\ & \{j \in \{1\} | false\} \\ \equiv & \\ & \emptyset \end{aligned}$$

In Definition 5.1 it has to hold for every $\overrightarrow{d}$, however we don't want to inspect all possible $\overrightarrow{d}$'s. We want to inspect it for a specific instantiation LPS of a $X(\overrightarrow{v_0})$.

5.2 Proposal 2 - without free variables

In Section 5.1 we see that Definition 5.1 can be improved, if we can give a specific $\overrightarrow{d}$. We try to improve that approximation, by giving $\overrightarrow{d}$ a specific value in S^2_L. So we define:

Definition 5.3. Let L be an LPS

$$S^2_L(\overrightarrow{d}) = \{j \in \{1, \ldots, n\} | \forall_{i \in I}(NF(c_i(\overrightarrow{d, e_i})) = \text{"}true\text{"}) \Rightarrow \overrightarrow{d}_{.j} = NF(\overrightarrow{g_i}(\overrightarrow{d, e_i})_{.j}))\}$$

We now can "instantiate" the S^2 with a $\overrightarrow{d}$. If we instantiate S^2 with $\overrightarrow{v_0}$, there is a problem. If we take a look at Example 5.4, we can see why:

Example 5.4. Let $E2$ be

```
proc P(x,y: Nat) = (x=0) -> P(x:= 1, y:= 0) +
                   (x=1) -> P(x:= x, y:= 1);
init P(x := 0, y:= 0);
```

If we use Definition 5.3, we get:

$$
\begin{aligned}
& S^2_{E2}(\langle 0,0\rangle) \\
\equiv & \\
& \{j \in \{1,2\} | \forall_{i \in I}(NF(c_i(\langle 0,0\rangle) = \text{"}true\text{"}) \Rightarrow \langle 0,0\rangle._j = NF(\overrightarrow{g_i}(\langle 0,0\rangle)._j))\} \\
\equiv & \\
& \{j \in \{1,2\} | \\
& \quad (NF(\overrightarrow{c_1}(\langle 0,0\rangle) = \text{"}true\text{"}) \Rightarrow \langle 0,0\rangle._j = NF(\overrightarrow{g_1}(\langle 0,0\rangle)._j)) \\
& \quad \wedge \\
& \quad (NF(\overrightarrow{c_2}(\langle 0,0\rangle) = \text{"}true\text{"}) \Rightarrow \langle 0,0\rangle._j = NF(\overrightarrow{g_2}(\langle 0,0\rangle)._j)) \\
& \} \\
\equiv & \\
& \{j \in \{1,2\} | \\
& \quad (\text{"}true\text{"} \Rightarrow \langle 0,0\rangle._j = NF(\overrightarrow{g_1}(\langle 0,0\rangle)._j)) \\
& \quad \wedge \\
& \quad (\text{"}false\text{"} \Rightarrow \langle 0,0\rangle._j = NF(\overrightarrow{g_2}(\langle 0,0\rangle)._j)) \\
& \} \\
\equiv & \\
& \{j \in \{1,2\} | \langle 0,0\rangle._j = NF(\overrightarrow{g_1}(\langle 0,0\rangle)._j)\} \\
\equiv & \\
& \{j \in \{1\} | ((0) = (1))\} \\
& \cup \\
& \{j \in \{2\} | ((0) = (0))\} \\
\equiv & \\
& \{2\}
\end{aligned}
$$

$S^2_{E2}(\langle 0,0\rangle) = \{2\}$. This indicates that the second process parameter is constant. However if we take another step from the initial state, we see the the that the second process parameter is not constant. We only inspect those states which are adjacent to the initial state, which might result in a wrong solution.

5.3 Proposal 3 - without free variables

To overcome the problem in which only the states adjacent to the initial state are compared, we suggest this proposal. First we redefine when the next state of summands should compared to the initial state. So if we have a condition and the condition is an open term, it might be that this condition will not reduce to a normal form which is either *true* or *false*. In such a case, there might be an valuation of the variables in the condition for which the condition holds. If this is the case we know the condition of a particular summand might be *true*, so the corresponding next state should be inspected. The summands for which the conditions are *false* are not inspected.

Definition 5.5 ($\perp$)**.** Let x be a process parameter. We define $\perp$ to be the value of process parameter x, if x is not constant.

We are interested in the largest subset of constant process parameters so:
Let $X \subseteq \{1, \ldots, n\}$ [2] be a set of indices of constant process parameters of an LPS:

[2] X is the set of indices which we are looking for

Definition 5.6. We define $\otimes_X(\overrightarrow{d})$:
Let $j \in \{1, \ldots, |\overrightarrow{d}|\}$.
If $j \notin X$ then $\otimes_X(\overrightarrow{d})_{.j} = \bot$.
If $j \in X$ then $\otimes_X(\overrightarrow{d})_{.j} = \overrightarrow{d}_{.j}$.

In order find the largest subset we give:

Definition 5.7. Let L be an LPS. We define: $S^3_L(\overrightarrow{v_0}) \subseteq \{1, \ldots, n\}$ as the biggest possible set such that:

$$\forall_{i \in I}(NF(c_i(\overrightarrow{\otimes_{S^3_L}(\overrightarrow{v_0}), e_i})) \neq \text{"}false\text{"} \Rightarrow \forall_{j \in S^3_L}(\overrightarrow{v_0}_{.j} = NF(\overrightarrow{g_i}(\overrightarrow{\otimes_{S^3_L}(\overrightarrow{v_0}), e_i})_{.j})))$$

Theorem 5.8. *Let L be an LPS and $\overrightarrow{d}$ a vector of process parameters*

$$S^3_L(\overrightarrow{d}) \subseteq S_L(\overrightarrow{d})$$

Definition 5.9 (@)**.** Let $\overrightarrow{d} = (d_1, \ldots d_n)$, let $j \in \{1, \ldots, |\overrightarrow{d}|\}$ and let x be a value.
$\overrightarrow{d}\,@_j(x)$ is defined as the substitution of the j^{th} element of $\overrightarrow{d}$ with the value of x.

Definition 5.10 ($\varphi(\overrightarrow{d}, X)$)**.** φ is a function of $\overrightarrow{D} \times \mathcal{P}(Nat) \to \mathcal{P}(\overrightarrow{D})$:

$$\varphi(\overrightarrow{d}, X) = \begin{cases} X = \emptyset & \to \{\overrightarrow{d}\} \\ X \neq \emptyset & \to \cup_{y \in \overrightarrow{D}_{.z}} \varphi(\overrightarrow{d}\,@_z(y), X \setminus \{z\}) \qquad \text{with } z \in X \end{cases}$$

Proof. 5.8

We do case distinction on $S^3_L(\overrightarrow{v_0})$:

- Let $S^3_L(\overrightarrow{v_0})$ be empty, then $S^3_L(\overrightarrow{v_0}) \subseteq S_L(\overrightarrow{v_0})$ holds trivially.
- Let $S^3_L(\overrightarrow{v_0})$ not be empty. We prove that if there is an element in $S^3_L(\overrightarrow{v_0})$, then this element is also in $S_L(\overrightarrow{v_0})$.
  $j \in S^3_L(\overrightarrow{v_0})$

  $\equiv$ { Def 5.7 of S^3_L}

  $j \in \{k \in \{1, \ldots, n\} | \forall_{i \in I}(NF(c_i(\overrightarrow{\otimes_{S^3_L}(\overrightarrow{v_0}), e_i})) \neq false \Rightarrow \overrightarrow{v_0}_{.k} = NF(\overrightarrow{g_i}(\otimes_{S^3_L}(\overrightarrow{v_0}), e_i)_{.k}))\}$

  $\equiv$ { Property of $\overrightarrow{\otimes_{S^3_L}(\overrightarrow{v_0}, e_i)}$}

  $j \in \{k \in \{1, \ldots, n\} | \forall_{i \in I, \overrightarrow{e_i} \in \overrightarrow{E_i}}(\forall_{\overrightarrow{x} \in \varphi(\overrightarrow{v_0}, S^3_L)}(c_i(\overrightarrow{x, e_i}) \Rightarrow \overrightarrow{v_0}_{.k} = \overrightarrow{g_i}(\overrightarrow{x, e_i})_{.k}))\}$

  $\Rightarrow$ $\{\varphi(\overrightarrow{v_0}, S^3_L) \supseteq R_L(\overrightarrow{v_0})$[3]$\}$

  $j \in \{k \in \{1, \ldots, n\} | \forall_{i \in I, \overrightarrow{e_i} \in \overrightarrow{E_i}}(\forall_{x \in R_L(\overrightarrow{v_0})}(c_i(\overrightarrow{x, e_i}) \Rightarrow \overrightarrow{v_0}_{.k} = \overrightarrow{g_i}(\overrightarrow{x, e_i})_{.k}))\}$

[3]See Example 5.11

$\equiv \quad \{\forall_{i\in I,\overrightarrow{e_i}\in\overrightarrow{E_i}}\forall_{x\in R_L(\overrightarrow{v_0})}c_i(x,e_i) \Rightarrow \overrightarrow{g_i}(\overrightarrow{x,e_i}) \in R_L(\overrightarrow{v_0}))\}$

$j \in \{k \in \{1,\ldots,n\} | \forall_{i\in I,\overrightarrow{e_i}\in\overrightarrow{E_i}}(\forall_{x\in R_L(\overrightarrow{v_0})}\forall_{y\in R_L(\overrightarrow{v_0})}(y = \overrightarrow{g_i}(x,e_i) \wedge c_i(x,e_i) \Rightarrow \overrightarrow{v_0}.k = y.k)\}$

$\equiv \quad \{\overrightarrow{v_0} \in R_L(\overrightarrow{v_0})\}$

$j \in \{k \in \{1,\ldots,n\} | \forall_{x\in R_L(\overrightarrow{v_0})}(\overrightarrow{x}.k = \overrightarrow{v_0}.k)\}$

$\equiv \quad \{$ Def 4.2 of $S_L\}$

$j \in S_L(\overrightarrow{v_0})$

Example 5.11. Let $S = \{1,2\}$. We have $\overrightarrow{z} = (1,5,3)$. The values of the 3^{th} vector can range between 1 and 4. If we want to map this vector to a statespace, knowing that the first and the second are constant is, we get the following state space:

$$\varphi(\overrightarrow{v_0}, S_L^3) = \{(1,5,1),(1,5,2),(1,5,3),(1,5,4)\}$$

which is a maximal statespace. If we have $R_L(\overrightarrow{z})$ (and we know this is reachable set is minimal) the maximal space space genarated would be $\{(1,5,1),(1,5,2),(1,5,3),(1,5,4)\}$ and a minimal set might be $\{(1,5,1),(1,5,4)\}$. So: $R_L(\overrightarrow{z}) \subseteq \{(1,5,1),(1,5,2),(1,5,3),(1,5,4)\} = (1,5,\perp) = \overrightarrow{z}$

So we have derived that each element in $S_L^3(\overrightarrow{v_0})$, is also in $S_L(\overrightarrow{v_0})$.
□

5.4 Proposal 4 - with free variables

We are interested in the largest subset of constant process parameters; for each process parameter it should hold that it should equal to all their next state process parameters.

Let $X \subseteq \{1, \ldots, n\}$ of constant process parameters of an LPS:
In order find the largest subset we give:

Definition 5.12. Let L be an LPS. We define S^4_L:

$S^4_L(\overrightarrow{v_0}) \subseteq \{1 \ldots n\}$

is the biggest subset, for which holds:

$$
\begin{aligned}
&\forall_{j \in S^L_4(\overrightarrow{v_0})}(\\
&\quad NF(\overrightarrow{v_0}.j) \notin FV \Rightarrow \\
&\quad \forall_{i \in I}(NF(c_i(\overrightarrow{\otimes^{FV}_{S^4_L}(\overrightarrow{v_0}), e_i})) \neq \text{false} \wedge \\
&\quad NF(\overrightarrow{g_i}(\overrightarrow{\otimes^{FV}_{S^4_L}(\overrightarrow{v_0}), e_i}).j) \notin FV \Rightarrow \\
&\qquad NF(\overrightarrow{g_i}(\overrightarrow{\otimes^{FV}_{S^4_L}(\overrightarrow{v_0}), e_i}).j) = NF(\overrightarrow{v_0}.j) \\
&\quad) \\
&\wedge \\
&\quad NF(\overrightarrow{v_0}.j) \in FV \Rightarrow \\
&\quad \exists_k(\forall_{i \in I}(NF(c_i(\overrightarrow{\otimes^{FV}_{S^4_L}(\overrightarrow{v_0}), e_i})) \neq \text{false} \wedge \\
&\quad NF(\overrightarrow{g_i}(\overrightarrow{\otimes^{FV}_{S^4_L}(\overrightarrow{v_0}), e_i}).j) \notin FV \Rightarrow \\
&\qquad NF(\overrightarrow{g_i}(\overrightarrow{\otimes^{FV}_{S^4_L}(\overrightarrow{v_0}), e_i}).j) = k) \\
&\quad) \\
&)
\end{aligned}
$$

If we compare Definition 5.12 to Definition 4.4 we see that Theorem 5.13 holds :

Theorem 5.13. *Let L be an LPS and $\overrightarrow{d}$ a vector op process parameters.*

$$S^4_L(\overrightarrow{d}) \subseteq S^{FV}_L(\overrightarrow{d})$$

6 Algorithm description

In this section we informally describe the algorithms for finding constant process parameters in LPSs with and without *free variables*. First a description is given about how to determine the constant process parameters when an LPS does not contain any free variables. Next an informal description is given for finding constant process parameters when an LPS contains free variables.

6.1 without free variables

First the *state vector* is constructed from the initial process of the linear process specification. Next the set S is defined, which S contains the indices for all process parameters that might be constant. It might be the case that all process parameters are constant so we start off with having all indices in set S. If a process parameter is detected to be variable, this process parameter is removed from the set S.

We keep on removing elements from S until no more elements can be removed. Removing the elements is done as follows: First we fill in all values of the process parameters, for which the index is in S, into the conditions. Next all conditions are rewritten and compared to *false*.

If the rewritten condition, in its normal form, is different from *false* the condition might be *true*, thus we calculate the nextstate of the current summand. We compare each element from the *state vector* with the nextstate vector for which the indices are in S. If an element is not equal, the corresponding index is removed from the set S. At value at the removed index of the *state vector* is replaced by $\perp$.

If no elements are removed from S during the iteration, there are either no more constant process parameters or all process parameters are constant. If this is the case we stop the iteration. Next all occurrences in of constant process parameters are substituted with their constant values and are constant process parameters are removed form the list of process parameters from the LPS.

6.2 with free variables

Before describing the algorithm informally we define some rules which should hold when we use *free variables*. The set S is again the set of indices of constant process parameters.

The values of process parameters can either be given a specific value or a *free variable*. In order to determine if a process parameter is constant or variable we define rules.

1. The process parameter has a specific value or is a *free variable* and a *free variable* is assigned to this process parameter:
   If *free variable* is assigned to a process parameter with a specific value, the value of this process parameter remains unchanged. If the index of process parameter is in S, it will not be removed from S. If we would allow replacement of a specific value with a *free variable*, this might lead to a fault result. Consider the next example; Let x_1 and x_2 be two specific values which are not equal to each other. Let v_1 be a *free variable*. Let the index of process parameter be in S. This process parameter has the value x_1 and is replaced with v_1, the index of process parameter stays in S. If in a next state this process parameter is substituted with x_2, the index remains in S. This indicates that the process parameter would remain constant. Clearly this observation is wrong. So we prohibited assigning *free variables* by specific values.

2. A process parameter which is a *free variable* is substituted with a specific value:
   The process parameter gets the value which is substituted and the process parameter remains in S if the specific value is not $\perp$. If the process parameter gets the value $\perp$ the process parameter is removed from S.

3. The process parameter which is specific value is substituted with another specific value:
   If the substituted value differs from the original value, the process parameter becomes marked variable and is removed from the set S. If they are equal, the process parameter remains in the set S if it already had been in the set S.

If no constant process parameters can be removed, the algorithm ends. However it is possible that we end in a situation like in Example 6.1. If we follow the algorithm, we see that the algorithm detects the 3^{th} element as a constant process parameter. However the 3^{th} is not a constant process parameter. To overcome the detection of fake constant process parameters, we have to take an extra pass on the LPS. The current state vector is compared to the nextstates of summands on which the condition is not false. If all process parameters of all nextstates

are equal[4] to the current state, then there are no fake constant process parameters. If the pass detects a process parameter which is not the same in the nextstate as in the current state, the algorithm starts over with:

- S without the indices of found fake constant process parameters.
- All the process parameters which are fake constant process parameters are substituted with the value $\perp$.

If the "Free Variable Checkup" does not find fake constants the algorithm ends.

Example 6.1. This LPS contains fake constant process parameters. FV_x indicates the x^{th} element in the set of FV.

act a;
proc X(a, b , c) = a. X(FV_4, FV_5, a) +
a. X(FV_6, FV_7, b) +
a. X(8 , 9, FV_8);
init X(FV_1, FV_2, FV_3);

In this part we only describe the difference between the algorithm without *free variables*. The algorithm remains basically the same. We only extend Algorithm Description 6.1 when comparing elements from the current state with the corresponding next state. When comparing we use the rules associated with *free variables*.

[4] with respect to the extended rules on *free variables*

7 Algorithms

7.1 without free variables

---

lpsconstelm (without free variables)

1: $curr := v_0$
2: $R := \{1 \ldots |v_0|\}$
3: $S := \emptyset$
4: **while** $S \neq R$ **do**
5: $\quad S := R$
6: $\quad$ **for all** $(i \in I) \wedge NF((c_i(curr)) \neq false)$ **do**
7: $\quad\quad$ **for all** $j \in S$ **do**
8: $\quad\quad\quad$ **if** $NF((g_i(curr).{}_j) \neq curr.{}_j)$ **then**
9: $\quad\quad\quad\quad curr.{}_j := \bot$
10: $\quad\quad\quad\quad R := R \setminus \{j\}$
11: $\quad\quad\quad$ **else if** $otherwise$ **then**
12: $\quad\quad\quad\quad skip$
13: $\quad\quad\quad$ **end if**
14: $\quad\quad$ **end for**
15: $\quad$ **end for**
16: **end while**
17: **return** S

7.2 with free variables

lpsconstelm (with free variables)

```
1:  curr := v_0
2:  R := {1 ... |v_0|}
3:  S := ∅
4:  while S = R do
5:    while S = R do
6:      S := R
7:      for all (i ∈ I) ∧ (c_i(curr) ≠ false) do
8:        for all j ∈ S do
9:          if curr.j ∈ FV then
10:           if g_i(curr).j ∉ FV then
11:             curr.j := g_i(curr).j
12:             if curr.j = ⊥ then
13:               R := R \ {j}
14:             end if
15:           else if g_i(curr).j ∈ FV then
16:             skip
17:           end if
18:         else if curr.j ∉ FV then
19:           if (g_i(curr).j ∉ FV) ∧ (g_i(curr).j ≠ curr.j) then
20:             curr.j = ⊥
21:             R := R \ {j}
22:           else if (g_i(curr).j = curr.j) ∨ (g_i(curr).j ∈ FV) then
23:             skip
24:           end if
25:         end if
26:       end for
27:     end for
28:   end while
      {Free Variable Checkup}
29:   for all j ∈ R do
30:     t := NILL
31:     for all i ∈ I do
32:       if g_i(curr).j ∉ FV ∧ t = NILL then
33:         t := g_i(curr).j
34:       else if g_i(curr).j ∉ FV ∧ t ≠ NILL ∧ t ≠ g_i(curr).j then
35:         R := R \ {j}
36:         curr.j := ⊥
37:       else if otherwise then
38:         skip
39:       end if
40:     end for
41:   end for
42: end while
43: return S
```

8 Test Cases

All specifications are given in mCRL2 specification. Transformation from a mCRL2 specification to an LPS file is done with the tool: *mcrl22lps*. Each transformation is executed with the `-no-cluster` option, unless mentioned otherwise.

Case 1

inputfile: `$DIR$/tests/lpsconstelm/case1.mcrl2`

info This case is designed to detect if the *lpsconstelm* can detect a simple constant process parameter.

---

act action: Nat;
proc P(i: Nat) = action(i).P(i);
init P(0);

Result All occurrences of i should be substituted by 0 and i should be removed from the list of process parameters.

Case 2

inputfile: `$DIR$/tests/lpsconstelm/case2.mcrl2`

info This case is designed to detect if the *lpsconstelm* can detect a simple non constant process parameter.

---

act action: Nat;
proc P(i: Nat) = action(i).P(i+1);
init P(0);

Result i is not constant, therefor i should not be substituted and removed. The LPS remains the same.

Case 3

inputfile: `$DIR$/tests/lpsconstelm/case3.mcrl2`

info This case is designed to detect if the *lpsconstelm* can detect a simple non constant and a simple constant process parameter.

---

act action: Nat;
proc P(i,j: Nat) = action(j).P(i+1, j);
init P(0,5);

Result i is not constant, i is not constant.i is not to be removed, occurrences of j should be substituted by 5 and j should be removed from the list of process parameters.

Case 4

inputfile: `$DIR$/tests/lpsconstelm/case4.mcrl2`

info This case is to designed to test the arguments `-no-condition`, `-no-reachable` and the the way the tool deals with multiple simple summands.

---

act action: Nat;
proc P(i,j: Nat) = true → action(j).P(i+1, j)+
false → action(j).P(i+1, j+1);
init P(0,5);

Result `-`: Only the fist summand is not false, therefor j is constant and i is not constant. All occurrences of j should be substituted by 5 and j should be removed from the list of process parameters. Only the first summand is written to the new LPS, because the second summand is never evaluated.

`-no-condition`: All the conditions of the summands are considered to be true. If this argument is used i and j are not constant[5] and therefore no process parameter should be substituted and to be removed.

`-no-reachable`: Only the fist summand is not false, therefor j is constant and i is not constant. All occurrences of j should be substituted by 5 and j should be removed from the list of process parameters. Both summands are written to in the new LPS, allthough the second summand is never inspected. Note that in the second summand j is the occurrences of j is substituted with 5.

Case 5

inputfile: `$DIR$/tests/lpsconstelm/case5.mcrl2`

info This case is designed to test the `-no-singleton` argument.

---

sort Singleton = struct x;
act action :Nat;
proc P(i : Nat, j : Singleton) = true → action(i). P(i+1,j);
init P(0,x);

Result `-`: j is found to be constant, so all occurrences of j are substituted by the value x

`-no-singleton` : "Singleton" has only one constructor. So all process parameters of type "Singleton" are not to be substituted and removed from the list of process parameters. So in this case no modification is made to the LPS.

Case 6

inputfile: `$DIR$/tests/lpsconstelm/case6.mcrl2`

info This test is designed to test a mCRL2 specification with multiple actions.
A multiple summand test.

---

act action :Nat;
proc P(i: Nat) = action(i). Q(i);
Q(i: Nat) = action(i). P(i);
init P(0);

Result From the generated LPS we see that i is constant. All occurrences of i are to substituted by 0 and i is removed from the list of process parameters.

Case 7

inputfile: `$DIR$/tests/lpsconstelm/case7.mcrl2`

into This case is designed to test if guards will influence the *lpsconstelm* correctly.

---

act action :Nat;
proc P(i,j: Nat) = (i > 5) → action(i). P(i+1,j) +
(i == 5) → action(j). Q(j);
Q(i: Nat) = action(i). Q(i);
init P(0,0);

Result

`-`:	If no argument is used, all the conditions are false. So $s3, j$ and i are marked constant. All occurrences are substituted and removed from the list of process parameters.
`-no-condition`:	If this argument is used only j will be marked constant. All occurrences of j are substituted by 0 and j is removed from the list of process parameters.

Generated LPS

```
var  freevar,freevar0: Nat;
proc P(s3: Pos, j,i: Nat) =
       (s3 == 2) ->
         action(i) .
         P(s3 := 2, j := freevar0)
     + (s3 == 1 && 5 < i ) ->
         action(i) .
         P(s3 := 1, i := i + 1)
     + (s3 == 1 && i == 5) ->
         action(j) .
         P(s3 := 2, j := freevar, i := j);

init P(s3 := 1, j := 0, i := 0);
```

Case 8

inputfile: `$DIR$/tests/lpsconstelm/case8.mcrl2`

info This case is designed to show the short comings of the algorithm. The *lpsconstelm* cannot detect equality of (i==5). It will detect that i does not stay constant.

act action: Nat;

proc X(i: Nat) = (i < 5) → action(i).X(i+1) +
(i == 5) → action(i).Y(i, i);
Y(i,j: Nat) = action(j).Y(i,j+1);

init X(0);

Result All process parameters are not constant, so none of the occurrences of a process parameter are substituted by their value and are removed from the list of process parameters.

Generated LPS

```
var  freevar0: Nat;
proc P(s3: Pos, i,j: Nat) =
       (s3 == 2) ->
         action(j) .
         P(s3 := 2, j := j + 1)
     + (s3 == 1 && i < 5) ->
         action(i) .
          P(s3 := 1, i := i + 1, j := freevar0)
     + (s3 == 1 && i == 5) ->
         action(i) .
         P(s3 := 2, j := i);

var  freevar: Nat;
init P(s3 := 1, i := 0, j := freevar);
```

References

[1] unknown author
Article not ready at the moment,

[2] J.W. Wesselink, *http://www.win.tue.nl/ wieger/mcrl2/html/index.html*
A C++ wrapper for the ATerm library.

[3] Aad Mathijssen
https://svn.win.tue.nl/viewcvs-checkout/MCRL2/trunk/specs/mcrl2.internal.txt, A description of the internal format of the mCRL2 language.

[4] unknown author, *Article not ready at the moment*