lpsparelm

F.P.M.Stappers

April 26, 2024

Abstract

This documentation describes the implementation and test cases of the tool *lpsparelm* in the mCRL2 toolset. Basically, *lpsparelm* is a tool which removes unused process parameter and unused sum variables from a linear process specification (LPS).

Contents

1 Introduction

The *lpsparelm* 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*. The filter removes process parameters and sum variables, which are not used, from the linear process specification. After the algorithm (Section 5) terminates, *lpsparelm* 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)

$$X(\overrightarrow{d:D}) = \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \overrightarrow{c_i}(\overrightarrow{d,e_i}) \to (a_i^1(\overrightarrow{f_{i,1}}(\overrightarrow{d,e_i}))|\ldots|a_i^{n(i)}(\overrightarrow{f_{i,n(i)}}(\overrightarrow{d,e_i})))\lhd t_i(\overrightarrow{d,e_i}) \cdot X(\overrightarrow{g_i}(\overrightarrow{d,e_i}))+$$

$$\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \overrightarrow{c_j}(\overrightarrow{d,e_j}) \to (a_j^1(\overrightarrow{f_{j,1}}(\overrightarrow{d,e_j}))|\ldots|a_j^{n(j)}(\overrightarrow{f_{j,n(j)}}(\overrightarrow{d,e_j})))\lhd t_j(\overrightarrow{d,e_j})+$$

$\sum_{\overrightarrow{e_\delta}:\overrightarrow{E_\delta}} \overrightarrow{c_\delta}(\overrightarrow{d, e_\delta}) \rightarrow \delta^{\triangleleft}\ t_\delta(\overrightarrow{d, e_\delta})$

Where I and J are disjoint.

If we speak about an LPS in this article we refer to Definition 2. The different states of the process are represented by the data vector parameter $\overrightarrow{d} : \overrightarrow{D}$. $\overrightarrow{D}$ may be a Cartesian product of n data types, meaning that $\overrightarrow{d}$ consist of a tuple $(d_1, \ldots, d_n)$. The LPS expresses that in state $\overrightarrow{d}$ it performs (multi)actions $a_i^1, \ldots, a_i^{n(i)}$, carrying data parameters $\overrightarrow{f_{i,1}}(\overrightarrow{d, e_i}), \ldots, \overrightarrow{f_{i,n(i)}}(\overrightarrow{d, e_i})$ and 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 $\overrightarrow{g_i} : \overrightarrow{D} \times \overrightarrow{E_i} \rightarrow \overrightarrow{D}$ and a function $c_i : \overrightarrow{D} \times \overrightarrow{E_i} \rightarrow \mathbb{B}$. Data parameters $\overrightarrow{e_i : E_i}$ are sum variables. These variables are used to action range over a data domain.

For an more detailed explanation of linear process specifications we refer to [1].

3 Definition lpsparelm

A parameter of an LPS which has no influence on condition, action arguments and time is removed from the LPS in the *lpsparelm* filter. Elimination of parameters can lead to a reduction when generating a state space of an LPS. If a process parameter ranges over an infinite domain, the number of generated states can even be reduced to a finite domain, using this operation.

Definition 3.1. Let $\overrightarrow{x}$ be a vector and n be $|\overrightarrow{x}|$. If we want to address the j^{th} element $(1 \leq j \leq |\overrightarrow{x}|)$ of vector $\overrightarrow{x}$, the do so by: $\overrightarrow{x}_{\cdot j}$

Definition 3.2. Let $w \in I \cup J$, where I and J are the set of summand indices; We define $\overrightarrow{vect^w} = (\overrightarrow{d, e_w})$
where $k = |\overrightarrow{d}|$ and $z = |\overrightarrow{e_w}|$. The length of $\overrightarrow{vect^w}$ is equal to $k + z$.

Definition 3.3 ($\boxtimes_X^w$)**.** Let $w \in W$, where W is a set of summand indices.
We define $\boxtimes_X^w$:
Let $j \in \{1, \ldots, |\overrightarrow{vect^w}|\}$.
If $j \notin X$ then $\boxtimes_X^w{}_{\cdot j} = y$ where $y \in \overrightarrow{D}_{\cdot j}$
If $j \in X$ then $\boxtimes_X^w{}_{\cdot j} = \overrightarrow{vect^w}_{\cdot j}$

Definition 3.4. Let M and L be an LPS, with $S_L \subseteq \{1, \ldots, n\}$ as the biggest possible set such that: $M \overset{\leftrightarrow}{=} L$, with M defined as follows:

$$
\begin{aligned}
&Y(\overrightarrow{d : D}) = \\
&\quad \sum_{k \in I} \sum_{\overrightarrow{e_k : E_k}} c_k(\boxtimes_{S_L}^k) \rightarrow a_k^1(\overrightarrow{f_{k,1}}(\boxtimes_{S_L}^k)) | \ldots | a_k^{n(k)}(\overrightarrow{f_{k,n(k)}}(\boxtimes_{S_L}^k)^{\triangleleft}\ t_k(\boxtimes_{S_L}^k) \cdot Y(\overrightarrow{g_k}(\boxtimes_{S_L}^k)) \\
+ \quad &\sum_{l \in J} \sum_{\overrightarrow{e_l : E_L}} c_l(\boxtimes_{S_L}^l) \rightarrow a_l^1(\overrightarrow{f_{l,1}}(\boxtimes_{S_L}^l)) | \ldots | a_l^{n(l)}(\overrightarrow{f_{l,n(l)}}(\boxtimes_{S_L}^l)^{\triangleleft}\ t_l(\boxtimes_{S_L}^l) \\
+ \quad &\sum_{\overrightarrow{e_\delta : E_{\delta^1}}} c_\delta(\boxtimes_{S_L}^\delta) \rightarrow \delta^{\triangleleft}\ t_\delta(\boxtimes_{S_L}^\delta)
\end{aligned}
$$

4 Description

It is possible having process parameters which can have any value without influencing the behavior of the LPS. If we look at the next states in an LPS in the LPS-format, we see that the

LPS only contains process parameters which influence the behavior and it omits those process parameters which do not influence the behavior. So, if process parameters do not occur in condition, action arguments or time tags, they do not influence the behavior in a direct way. If we collect all process parameters, which are used in the condition, action arguments and time tags, we know which process parameters are used. To indicate which process parameters are used, we keep a set $UsedPP$[1] which contains all indices used process parameters. To find all process parametersin a condition, action argument or time tag we have function called $FindUsedPP$. This function returns a set of process parameter indices which occur in a given condition, action argument or time tag.

A next state is always used as an input state, to calculate another next state. So for each process parameter we have to check while all process parameters which are used to construct a next state are in the set of used process parameters. If a process parameter is used as an input and is not in the set of used process parameters this process parameter is added to the set of indices. Also, we have to check, that all process parameters on which the added process parameter depends, are also in the list of used process parameters. We continue adding process parameters dependant on other process parameters which are also in the set of used process parameter indices.

In short: Throughout the run of the algorithm process parameters get marked if they are used in an action, time or condition. Process parameters which are dependant of other marked process parameters also get marked. If a process parameter gets marked it should not be removed. Initially, all parameters in the LPS are unmarked. After the algorithm terminates all unmarked process parameters are removed. A process parameter gets marked if it occurs in one of the following places:

- In a condition: $c_i(\overrightarrow{vect^i})$ for some $i \in I \cup J$
- In an action argument: $(\overrightarrow{f_{i,j}}(\overrightarrow{vect^i})$ for some $i \in I \cup J$ and $j \in \{1, \ldots, n\}$
- In a time tag: $t_i(\overrightarrow{vect^i})$ for some $i \in I$
- In an argument: $\overrightarrow{g_i}(\overrightarrow{vect^i})_j$ for some $i \in I \cup J$ and $j \in \{1, \ldots, n\}$ where the j-th process parameter is marked.

[1] $UsedPP = S_M$: Due the length off the word $UsedPP$ the Definition 3.4 would get to long and clipped

5 Algorithm

Algorithm 1 lpsparelm

1: $UsedPP := \emptyset$
2: **for all** $i \in I \cup J$ **do**
3: $\quad UsedPP := UsedPP \cup FindUsedPP(\overrightarrow{c_i}(\overrightarrow{vect^i}))$
4: $\quad$ **for** $j := 1$ to $n(i)$ **do**
5: $\quad\quad UsedPP := UsedPP \cup FindUsedPP(\overrightarrow{f_{i,j}}(\overrightarrow{vect^i})$
6: $\quad$ **end for**
7: $\quad UsedPP := UsedPP \cup FindUsedPP(\overrightarrow{t_i}(\overrightarrow{vect^i}))$
8: **end for**
9: $n := 0$
10: **while** $|UsedPP| > n$ **do**
11: $\quad n := |UsedPP|$
12: $\quad$ **for all** $i \in I$ **do**
13: $\quad\quad$ **for all** $j \in UsedPP$ **do**
14: $\quad\quad\quad UsedPP := UsedPP \cup FindUsedPP(\overrightarrow{g_i}(\overrightarrow{vect^i})_{.j})$
15: $\quad\quad$ **end for**
16: $\quad$ **end for**
17: **end while**
18: **return** $UsedPP$

$UsedPP$	:	The set of indices of used process parameters.
$FindUsedPP$	:	A routine which returns a set of indices of used process parameters, which occur in a given argument.

6 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/lpsparelm/case1.mcrl2`

info This test case will show if if the *lpsparelm* works without condition, action and time arguments.

---

act a;
proc X(i:Nat) = a.X(i)
init X(2);

Result Process parameter i should be removed.

Case 2

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

info This test case will show if if the *lpsparelm* works with an action argument.

---

act a: Nat;
proc X(i,j:Nat) = a(i). X(i,j)
init X(0,1);

Result Process parameter j should be removed.

Case 3

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

info This test case will show if if the *lpsparelm* works with a condition argument.

---

act a;
proc X(i,j:Nat) = (i == 5) $\rightarrow$ a. X(i,j)
init X(0,1);

Result Process parameter j should be removed.

Case 4

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

info This test case will show if if the *lpsparelm* works with a time argument.

---

act a;
proc X(i,j:Nat) = a@i. X(i,j)
init X(0,4);

Result Process parameter j should be removed.

Case 5

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

info This test case will show if the *lpsparelm* checks for dependant process parameters.

---

act a: Nat ;
act b;
proc X(i,j,k:Nat) = a(i).X(k,j,k) +
b.X(j,j,k);
init X(1,2,3);

Result No process parameters should be removed.

Case 6

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

info This test case will show how the tool reacts on free variables and multiple summands.

---

act act1, act2, act3: Nat;
proc X(i: Nat) = (i < 5) → act1(i).X(i+1) +
(i == 5) → act3(i).Y(i, i);
Y(i,j: Nat) = act2(j).Y(i,j+1);
init X(0);

Result No process parameter should be removed.
Note: Different linearization might lead to other results.

Genarated LPS

```
var  freevar0: Nat;
proc P(s3: Pos, i,j: Nat)
 (s3 == 1 && i < 5) ->
   act1(i) .
   P(s3 := 1, i := i + 1, j := freevar0)
+ (s3 == 1 && i == 5) ->
   act3(i) .
   P(s3 := 2, j := i)
+ (s3 == 2) ->
   act2(j) .
   P(s3 := 2, j := j + 1);
```

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

Case 7

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

info This test case will show how the tool reacts on time, actions, conditions, multiple summands and dependant process parameters.

---

act act1, act2, act3: Nat;
proc X(i,z,j: Nat) = (i < 5) → act1(i)@z.X(i+1,z, j) +
(i == 5) → act3(i).X(i, j, 4);
init X(0,5, 1);

Result No process parameter should be removed.

Genarated LPS

```
proc P(i,z,j: Nat) =
      (i == 5) ->
        act3(i) .
        P(z := j,j := 4)
    + (i < 5) ->
        act1(i) @ Nat2Real(z) .
        P(i := i + 1);

init P(i := 0, z := Pos2Nat(5), j := Pos2Nat(1));
```

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://github.com/mCRL2org/mCRL2/blob/master/doc/specs/mcrl2.internal.txt, A description of the internal format of the mCRL2 language.