LPE determinism

Djurre van der Wal

April 29, 2019

Contents

Chapter 1

Definitions

1.1 Restricted LPE form

LPEs are considered to be in *restricted LPE form*, a subset of *LPE form* in which hidden variables are not permitted and in which there must be exactly one *ChanOffer* per summand. The grammar of the restricted LPE form is

$$
\begin{aligned}
\mathit{LPE} &::= \texttt{ProcDef}\ [C_1 :: K_1, \cdots, C_n :: K_n]\ [p_1 :: T_1, \cdots, p_k :: T_k]\ \mathit{Body} \\
\mathit{Body} &::= \texttt{Choice}\ [\,\mathit{ActionSmd}, \cdots, \mathit{ActionSmd}\,] \\
\mathit{ActionSmd} &::= \texttt{ActionPref}\ \mathit{ActOffer}\ \mathit{ProcInst} \\
\mathit{ActOffer} &::= \texttt{ActOffer}\ \mathit{ChanOffers}\ [\,]\ \mathit{VExpr} \\
\mathit{ChanOffers} &::= [\,\mathit{ChanOffer}\,] \\
\mathit{ChanOffer} &::= \mathit{ChanId}\ \ [\mathit{ChanFlag}, \cdots, \mathit{ChanFlag}]\ \ [\texttt{Quest}\ x_1, \cdots, \texttt{Quest}\ x_m] \\
\mathit{ChanId} &::= C_1 \mid \cdots \mid C_n \\
\mathit{ChanFlag} &::= \texttt{INVISIBLE} \mid \texttt{CONFLUENT} \mid \texttt{QUIESCENT} \\
\mathit{ProcInst} &::= \texttt{ProcInst}\ \mathbf{P}\ \ [\,C_1, \cdots, C_n\,]\ [\,\mathit{VExpr}, \cdots, \mathit{VExpr}\,]
\end{aligned}
$$

with the following restrictions:

- *Body* should comply with traditional TorXakis requirements. More precisely, the communication variables $\{x_1, \cdots, x_m\}$ must match the signature of the *ChanId* channel that occurs in the same rule; and the number of *VExpr*s in the *ProcInst* rule must be equal to k and their sorts must match $[T_1, \cdots, T_k]$.
- Parameters $p_1, \cdots p_n$ as well as the communication variables $x_1, \cdots, x_m$ of all *ActionSmd*s must be unique across the process.

1.2 LPE elements

The following symbols are used to denote the elements of a process $\mathbf{P}$ that is in restricted LPE form:

- SMDS($\mathbf{P}$) is a list that contains all summands of $\mathbf{P}$;
- s_i denotes the ith summand in SMDS($\mathbf{P}$);
- Σ is the set of all channels that are used by the summands of $\mathbf{P}$;
- k denotes the number of data parameters that $\mathbf{P}$ requires;
- p_i denotes the ith data parameter that $\mathbf{P}$ requires;
- $P = \{\ p_j \mid 1 \leq j \leq k\ \}$ is the set of all data parameters of $\mathbf{P}$;
- $v_0(p)$ is a closed expression that defines the initial value of each data parameter $p \in P$.

The following definition is used to formally reference the elements of summand s_i, the ith summand in SMDS($\mathbf{P}$):

$$s_i = C_i \texttt{ ? } x_i(1) \cdots \texttt{ ? } x_i(m_i)\ [[g_i]]\ \texttt{>->}\ P(v_i(p_1),\ \cdots,\ v_i(p_k))$$

where

- $C_i \in \Sigma$ is the name of the channel over which summand s_i communicates (channel flags have been encoded into the channel name);
- $m_i \geq 0$ is the number of variables that summand s_i uses to communicate over channel C_i;
- $x_i(j)$ is the jth variable that summand s_i uses locally (communication variables first, followed by hidden variables);
- g_i is the guard of summand s_i (the only free variables in this expression must be elements in $P \cup \{\ x_i(j) \mid 1 \leq j \leq m_i\ \}$);
- $v_i(p)$ is an expression that defines the new value of data parameter $p \in P$ after the application of summand s_i (the only free variables in this expression must be elements in $P \cup \{\ x_i(j) \mid 1 \leq j \leq m_i\ \}$).

Note that, because $\mathbf{P}$ is in restricted LPE form, the following assumption generally holds for two summands, s_α and s_β:

$$C_\alpha = C_\beta \longrightarrow m_\alpha = m_\beta$$

1.3 Starter summands

Let summand $s_\alpha \in \text{SMDS}(\mathbf{P})$, referencing its elements conform 1.2. Summand s_α is said to be a *starter summand* of $\mathbf{P}$ if the following expression is satisfiable:

$$g_\alpha[p \to v_0(p) \mid p \in P]$$

(We rely on the assumption that the guard of each summand of $\mathbf{P}$ is satisfiable or unsatisfiable when $\mathbf{P}$ is in a fully specified state; instead, we could over-approximate, and also consider summands for which the expression *could be* satisfiable to be starter summands.)

The set of all starter summands of LPE $\mathbf{P}$ is denoted by

$$\text{STARTERS}(\mathbf{P}) = \{\ s_\alpha \mid s_\alpha \text{ is a starter summand of } \mathbf{P}\ \}$$

For convenience, we write starter summands filtered by their channel as

$$\text{STARTERS}_C(\mathbf{P}) = \{\ s_\alpha \mid s_\alpha \in \text{STARTERS}(\mathbf{P}),\ C_\alpha = C\ \}$$

1.4 Possible successors

Let $s_\alpha,\ s_\beta \in \text{SMDS}(\mathbf{P})$, referencing their elements conform 1.2. Summand s_α is said to be a *possible successor* of s_β if the following expression *could be* satisfiable:

$$g_\alpha[p \to v_\beta(p) \mid p \in P] \wedge g_\beta$$

(We cannot rely on the assumption that the guard of each summand of $\mathbf{P}$ is satisfiable or unsatisfiable when $\mathbf{P}$ is in a fully specified state because the current state is only symbolically available.)

Let $s_\alpha \in \text{SMDS}(\mathbf{P})$ be a summand. The set of all possible successors of s_α is denoted by

$$\text{PSUCCS}(s_\alpha) = \{\ s_\beta \mid s_\beta \in \text{SMDS}(\mathbf{P}) \text{ is a possible successor of } s_\alpha\ \}$$

For convenience, we write possible successors filtered by their channel as

$$\text{PSUCCS}_C(s_\alpha) = \{\ s_\beta \mid s_\beta \in \text{PSUCCS}(s_\alpha),\ C_\beta = C\ \}$$

1.5 Summand non-determinism

Informally, two summands are considered *non-deterministic* if they can cause the same action to be enabled under the same circumstances, and if the action could lead to different next states. The words 'can' and 'could' are chosen on purpose, because proving that two summands meet these requirements can be prohibitive. In order to guarantee that all non-determinism is removed from the LPE, it must be assumed that two summands are non-deterministic unless it can be proven otherwise. This means that two summands are considered non-deterministic if and only if they are not *deterministic*.
Consider two summands, s_α and s_β, and reference their elements conform 1.2. Summands s_α and s_β are said to be *deterministic* if one (or more) of these conditions holds:

- s_α and s_β are the same summand; that is, $s_\alpha = s_\beta$.

  (This is *not* a sufficient condition for determinism if a summand would have hidden variables!)

- s_α and s_β communicate over different channels; that is, $C_\alpha \neq C_\beta$.

- s_α and s_β cannot be enabled under the same circumstances, which is the case if $\neg(g_\alpha \wedge g_\beta)$ is a tautology.

- s_α and s_β always lead to different next states, which is the case if

  $$g_\alpha \wedge g_\beta \rightarrow \bigwedge_{j=1}^{k} v_\alpha(p_j) = v_\beta(p_j)$$

  is a tautology.

  (Note that this equation also captures the previous condition.)

Using the now established notion of non-determinism, we define the predicate

$$\text{NONDET}(S) = \forall s_\alpha,\ s_\beta \in S\ .\ s_\alpha \text{ and } s_\beta \text{ are non-deterministic}$$

to express whether a set of summands S is non-deterministic.

1.6 Enabled-starter signature

The *enabled-starter signature* of a process $\mathbf{P}$ in restricted LPE form is defined as

$$\mathrm{ESS}(\mathbf{P}) = \bigcup_{C \in \Sigma} \{\ (C,\ S,\ S') \mid \begin{array}{c} S \cup S' = \textsc{starters}_C(\mathbf{P}), \\ S \cap S' = \emptyset,\ S \neq \emptyset, \\ |S| = 1 \vee \textsc{nondet}(S) \end{array} \ \}$$

Note that the computation of the enabled-starter signature has a complexity that is exponential.

The fact that $(C,\ S,\ S') \in \mathrm{ESS}(\mathbf{P})$ implies that there may exist a state in which

- $\mathbf{P}$ can communicate via the channel C by using a summand in S;
- all summands in S are enabled; and
- all summands in S' are disabled.

1.7 Enabled-successor signature

The *enabled-successor signature* of a summand s_α is defined as

$$\mathrm{ESS}(s_\alpha) = \bigcup_{C \in \Sigma} \{\ (C,\ S,\ S') \mid \begin{array}{c} S \cup S' = \textsc{psuccs}_C(s_\alpha), \\ S \cap S' = \emptyset,\ S \neq \emptyset, \\ |S| = 1 \vee \textsc{nondet}(S) \end{array} \ \}$$

Note that the computation of the enabled-successor signature has a complexity that is exponential.

The interpretation of $(C,\ S,\ S') \in \mathrm{ESS}(s_\alpha)$ is analogous to the interpretation of $(C,\ S,\ S') \in \mathrm{ESS}(\mathbf{P})$ in Section 1.6.

1.8 Next-ESS function

Let $(C,\ S,\ S') \in \lambda$ where λ is some signature from Section 1.6 or Section 1.7. Tuple $(C,\ S,\ S')$ contains sufficient information to symbolically compute the signature that results after the application of an arbitrary summand from S:

$$\textsc{next}((C,\ S,\ S')) = \bigcup_{s_\alpha \in S} \mathrm{ESS}(s_\alpha)$$

Chapter 2

Algorithm

2.1 Intuition

An enabled-starter/summand signature (ESS) of a non-deterministic LPE gives us an over-approximation of the summands that could be enabled after initialization or after a transition (see Sections 1.6 and 1.7). This means that an ESS symbolically represents a set of states that are modeled by the LPE.
An ESS also gives us sufficient information to symbolically compute the ESSs of successor summands (see Section 1.8), which means that an ESS can be computed for every (reachable) state that the LPE models using fixed-point iteration. Unsurprisingly, this is the first step of the algorithm.
Next, the algorithm associates each computed ESS with a unique number. It then creates a new LPE with the same signature as the non-deterministic LPE, but with a newly added data parameter y. By design of the algorithm, each summand of the new LPE – although based on a summand from the non-deterministic LPE – is guaranteed to only be enabled if y is equal to the number that identifies a specific ESS. Consequently, if all summands of the new LPE that require the same value of y are deterministic, the new LPE is also deterministic.
Each tuple in an ESS is disjoint from all other tuples because either the channel is different or there is a summand that is enabled in one tuple and disabled in the other. The algorithm constructs guards that reflect this. However, the algorithm constructs a summand for each summand that is enabled according to a tuple.

2.2 Instructions

The algorithm consists of two main phases: the computation of the enabled-starter/summand signatures of the non-deterministic LPE, and the construction of a new, deterministic LPE by using those signatures.
Do the following:

1. Compute

$$\Lambda_0 = \{\ \mathrm{ESS}(\mathbf{P})\ \} \cup \{\ \mathrm{ESS}(s_\alpha) \mid s_\alpha \in \textsc{smds}(\mathbf{P})\ \}$$

which is the enabled-starter signature combined with the enabled-successor signatures of all summands.

2. Compute

$$\Lambda_n = \Lambda_{n-1} \cup \{\ \textsc{next}(\phi) \mid \phi \in \textsc{flatten}(\Lambda_{n-1})\ \}$$

where

$$\textsc{flatten}(\Lambda) = \bigcup_{\lambda \in \Lambda} \lambda$$

for a value of $n > 0$ that is large enough so that $\Lambda_n = \Lambda_{n-1}$. Λ_n contains Λ_0 and all signatures that are reachable from Λ_0.

3. If

$$\forall \lambda \in \Lambda_n,\ C \in \Sigma\ .\ |\{\ (C,\ S,\ S') \mid (C',\ S,\ S') \in \lambda,\ C' = C\ \}| = 1$$

then the LPE is already deterministic, and the remainder of the algorithm can be skipped.

4. Compute

$$\textsc{before}(\lambda) = \{\ \lambda' \mid \lambda' \in \Lambda_n,\ \theta \in \lambda',\ \textsc{next}(\theta) = \lambda\ \}$$

for all $\lambda \in \Lambda_n$.

5. Compute

$$\textsc{pids} = \{\ \textsc{pids}_\lambda(\theta) \mid \lambda \in \Lambda_n,\ \theta \in \lambda\ \}$$

where

$$\textsc{pids}_\lambda((C,\ S,\ S')) = \{\ (C,\ s,\ S,\ S',\ \lambda) \mid s \in S\ \}$$

6. Create a new LPE with the same signature as the original LPE except that the data parameters are as follows:

   - The new LPE has the data parameters of the original LPE;
   - The new LPE has an additional data parameter y of type $\mathbb{N}$;
   - The new LPE has an additional data parameter f^π of type $\mathbb{B}$ for all $\pi \in \textsc{pids}$;
   - The new LPE has an additional data parameter p^π for all $p \in P$ and for all $\pi \in \textsc{pids}$, where the sort of p^π is the same as the sort of p.

Note that the number of data parameters of the new LPE relates exponentially to the number of data parameters of the original LPE.

7. Create an injection $q : \Lambda_n \to \mathbb{N}$.

8. Consider each $\lambda \in \Lambda_n$ and each $(C,\ S,\ S') \in \lambda$. For each $s \in \text{BEFORE}(\lambda)$, create a new summand as follows:

$$C \mathbin{?} x_\alpha(1) \ \cdots \ \mathbin{?} x_\alpha(m_\alpha)\ [[G]] \mathbin{>\!\!\to} \mathbf{P}(\cdots)$$

where G is defined as

$$y = q(\lambda)\ \wedge\ \forall s_i \in S\ .\ g_i[X_i]\ \wedge\ \forall s_i \in S'\ .\ \neg g_i[X_i]$$

and where

$$X_i = [\ x_i(j) \mapsto x_\alpha(j) \mid 1 \leq j \leq m_\alpha\ ]$$

Data parameters of the new summand are assigned in the following way:

$$\begin{aligned}
y &:= q(\text{NEXT}(\lambda)) \\
p_1 &:= p_1,\ \cdots,\ p_k := p_k \\
f_\pi &:= \textbf{true} \text{ for all } \pi \in \text{PIDS}_\lambda((C,\ S,\ S')) \\
f_\pi &:= \textbf{false} \text{ for all } \pi \in \text{PIDS} \setminus \text{PIDS}_\lambda((C,\ S,\ S')) \\
p_1^\pi &:= v_s(p_1),\ \cdots,\ p_k^\pi := v_s(p_k) \text{ for all } \pi = (C,\ s,\ S,\ S',\ \lambda) \in \text{PIDS}_\lambda((C,\ S,\ S')) \\
p_1^\pi &:= v_s(p_1)[X_\pi],\ \cdots,\ p_k^\pi := v_s(p_k)[X_\pi] \text{ for all } \pi \in \text{PIDS} \setminus \text{PIDS}_\lambda((C,\ S,\ S'))
\end{aligned}$$