Linearization of Hybrid Chi Using Program Counters

U.Khadim, D.A. van Beek, P.J.L. Cuijpers
{u.khadim, d.a.v.beek, p.j.l.cuijpers}@tue.nl

Department of Mathematics and Computer Science
Department of Mechanical Engineering
Overview

• What is Linearization?
• A Linear Process Equation. Data structures in an LPE. Size of an LPE.
• Input to our algorithm
• Output of our algorithm
• The Linearization Algorithm.
  • Sequential Composition
  • Examples
  • Parallel Composition
  • General Observations
• Conclusion
What is Linearization?

Elimination theorems in $ACP$ style process algebras.

All closed process terms can be rewritten into a basic term.

Basic term-atomic actions, basic operators. No parallel operator.

Advantages of Elimination. Proofs become simpler.

Linearization of $\mu$CRL, HyPA, Hybrid Chi. Linearization of Hybrid Chi by R.J.M. Theunissen
The basic term in $\mu$CRL is a linear process equation (LPE).

- Complete system specification in a single recursive equation.
- It resembles a right linear data parameterized grammar.
Linear Process Equation

\[ X(d : D) = \sum_{i \in I} \sum_{e \in E_i} a_i \cdot X(g_i(d, e)) < c_i(d, e) \triangleright \delta \]

\[ + \sum_{j \in I} \sum_{e \in E_j} a_j < c_j(d, e) \triangleright \delta \]

- \( I, J \): some index sets
- \( a_i, a_j \): atomic actions
- \( d, e \): data values
- \( D, E_i, E_j \): data types
- \( c_i, c_j \): Conditions on data values
- \( g_i(d, e) \): new data values for the next recursive call
- \( \sum \): indicates there can be more than one option
Use of Data Structures

Some data structure to simulate the behaviour of various compositions, e.g. sequential, alternative, parallel compositions.

- $\mu$ CRL uses stacks and “lists of multi-lists”.
- HyPA uses stacks
- Previous linearization of Hybrid Chi does not use any data structures.
- We use counters that we call program counters.

The choice of a data structure also determines what subset of the process language can be linearized.
Recursive occurrences of parallel composition

\[ X = (a.X \parallel b) \]
\[ (a.(a.X \parallel b) \parallel b) \]
\[ (a.(a.(a.X \parallel b) \parallel b) \parallel b) \]

We do not allow such processes in our algorithm.

How many program counters are needed to linearize X?
Size of an LPE

- Eliminating a parallel operator from a process term increases its size.
- Semantics of a parallel operator includes communication and interleaving.

\[
a; b; c; d \parallel e; f; g; h
\]

can perform actions in order:

\[
abcdefgh \quad eabcdefgh \quad \ldots
\]

\[
abefghcd \quad aebcdghtgh \quad \ldots
\]
We use discrete counters to model interleaving.

\[ X = (i_1 = 8) \rightarrow a, i_1 := 6; X \]
\[ \rightarrow (i_1 = 6) \rightarrow b, i_1 := 4; X \]
\[ \rightarrow (i_1 = 4) \rightarrow c, i_1 := 2; X \]
\[ \rightarrow (i_1 = 2 \land \neg \text{Odd}(\{i_2\})) \rightarrow d, i_1 := 1; X \]
\[ \rightarrow (i_1 = 2 \land \text{Odd}(\{i_2\})) \rightarrow d, i_1 := 0, i_2 := 0 \]
\[ \rightarrow (i_2 = 8) \rightarrow e, i_2 := 6; X \]
\[ \rightarrow (i_2 = 6) \rightarrow f, i_2 := 4; X \]
\[ \rightarrow (i_2 = 4) \rightarrow g, i_2 := 2; X \]
\[ \rightarrow (i_2 = 2 \land \neg \text{Odd}(\{i_2\})) \rightarrow h, i_2 := 1; X \]
\[ \rightarrow (i_2 = 2 \land \text{Odd}(\{i_2\})) \rightarrow h, i_2 := 0, i_1 := 0 \]
A subset of hybrid Chi Language $\mathcal{P}_s$, $p_s \in \mathcal{P}_s$:

\[
p_s ::= \ p_{\text{atom}} \quad \text{atomic actions}
\|
\ p_u \quad \text{invariant and urgency conditions}
\|
\ u \xrightarrow{\sim} p_s \quad \text{initialization}
\|
\ p_s; p_s \quad \text{sequential composition}
\|
\ p_s \mid p_s \quad \text{alternative composition}
\|
\ p_s \parallel p_s \quad \text{parallel composition}
\|
\ \partial_A(p_s) \quad \text{Encapsulation}
\|
\ \partial_H(p_s) \quad \text{Send and receive action encapsulation}
\]
$p_s \text{ continued..}$

| $\nu_H(p_s)$ | urgent channel communication |
| $[[H H :: p_s]]$ | channel scope $[[H]]$ |
| $p_R$ | restricted use of recursion |
| $[[V \sigma_\bot, C, L :: p_s]]$ | variable scope $[[V]]$ |
Input to the Algorithm

Normalize: \( \mathcal{P} \mapsto \tilde{\mathcal{P}} \)

- Syntax of process terms in set \( \mathcal{P} \), with \( p \in \mathcal{P} \):

\[
p ::= \begin{array}{c}
p_s \\
p_s; X_i \\
p_s; p \\
p \parallel p
\end{array}
\]

- No recursion variables in parallel composition.
- \( p_s \) contains parallel composition.
- Only tail recursion is allowed.
\( \tilde{p}, \tilde{q}, \tilde{r} \) denote output process terms.

\[
\text{Normalize}(p) = [v \sigma_{pc} \cup \sigma_{\perp}, C, L \\
:: [R \{X \mapsto \bar{p}\} :: u \land u_{pc} \rightsquigarrow X ]]
\]

If \( p \) does not contain any local variables then important features that are left behind:

\[
\text{Normalize}(p) = [v \sigma_{pc} \\
:: [R \{X \mapsto \bar{p}\} :: u_{pc} \rightsquigarrow X ]]
\]
\[
\text{Normalize}(p) = [v \sigma_{pc} \\
:: [R \{X \mapsto \overline{p}\} :: u_{pc} \bowtie X ]]
\]

- \(\sigma_{pc}\) declares program counters
- \(u_{pc}\) initializes program counters
- \(\{X \mapsto \overline{p}\}\) A single recursion definition
- \(\overline{p}\) An LPE
Syntax of the process terms in set $\overline{P}$:

$$
\overline{p} ::= \quad b_{pc} \rightarrow p_u
\mid \quad b_{pc} \rightarrow p_{act}, \text{update}(X_i); \quad X
\mid \quad b_{pc} \rightarrow p_{act}, \text{ap}_{pc}; \quad X
\mid \quad b_{pc} \rightarrow p_{act}, \text{ap}_{pc}
\mid \quad \overline{p} \parallel \overline{p}
$$
Sequential Composition $p; q$

$$\text{Normalize}(p; q) = \left\lfloor V \sigma_{pc} :: \right\rfloor_R \{ \begin{array}{c} \ X \mapsto \text{Setzero}(\text{FSequent} (\text{Incrpcs} (\bar{p}, \text{value}(u^q_{pc}, 1)), \bar{q}) \\ \ \qquad \qquad \qquad \qquad \qquad \downarrow \bar{q} \qquad \qquad \qquad \qquad \qquad \downarrow \bar{q} \} \\
\} :: u_{pc} \sim X \right\rfloor$$

- $\sigma_{pc}$ declares pcs
- $u_{pc}$ initializes pcs
- $\tilde{q} = \text{Normalize}(q)$
- $\bar{q}$ the LPE in Normalize($q$)
- $u^q_{pc}$ initializes pcs in Normalize($q$)
• Number of program counters is maximum of the number used in \( \text{Normalize}(p) \) and \( \text{Normalize}(q) \).

• Initialization predicate: \( u_{pc} = \text{Incrpcs}(u_{pc}^p, \text{value}(u_{pc}^q, 1)) \)
  
In case \( \text{pcs}(\text{Normalize}(q)) > \text{pcs}(\text{Normalize}(p)) \), deactivate extra program counters of \( \text{Normalize}(p) \).

• \( \text{Incrpcs} : (\mathcal{U}_{pc} \times \mathbb{N} \rightarrow \mathcal{U}_{pc}) \cup (\mathcal{P} \times \mathbb{N} \rightarrow \mathcal{P}) \)

\[
\text{Incrpcs}(x, n) = x[i_k = c_k + n/i_k = c_k]_{k \in \mathbb{N}}
\]

where \( c_k \) is a natural number
FSequent : $\overline{P} \times \overline{P} \rightarrow \overline{P}$:

- Initializes the program counters according to their initial values in Normalize($q$).
- If Normalize($q$) has local variables then initialize them.
- Deactivate extra program counters of Normalize($p$) if $\text{pcs}(\text{Normalize}(q)) > \text{pcs}(\text{Normalize}(p))$.
- Add a recursive call to $X$.
Examples

\[ p = a; b; c; d \]

\[ \widetilde{p} = \llbracket V \{ i_1 \mapsto \bot \} :: [R \{ X \mapsto \overline{p} \} :: (i_1 = 8) \leadsto X ] \rrbracket \]

\[ \overline{p} = (i_1 = 8) \rightarrow a, i_1 := 6; X \\
(i_1 = 6) \rightarrow b, i_1 := 4; X \\
(i_1 = 4) \rightarrow c, i_1 := 2; X \\
(i_1 = 2) \rightarrow d, i_1 := 0; X \]
\[ p = a; b \parallel c \]

\[ \tilde{p} = \llbracket v \{ i_1 \mapsto \bot \} \quad :: \quad [r \{ X \mapsto \tilde{p} \} :: (i_1 = 4) \bowtie X ] \rrbracket \]

\[ \bar{p} = \begin{cases} (i_1 = 4) \rightarrow a, i_1 := 2; X \\ (i_1 = 2) \rightarrow b, i_1 := 0 \\ (i_1 = 4) \rightarrow c, i_1 := 0 \end{cases} \]
\[ p = a; b \parallel c; d \]

\[ \tilde{p} = \left[ \forall \{ i_1 \mapsto \bot \} \right] \left[ \forall \{ X \mapsto \bar{p} \} :: (i_1 = 6) \right] X \]

\[ \bar{p} = (i_1 = 6) \rightarrow a, i_1 := 4; X \\
(i_1 = 4) \rightarrow b, i_1 := 0 \\
(i_1 = 6) \rightarrow c, i_1 := 2; X \\
(i_1 = 2) \rightarrow d, i_1 := 0 \]
Parallel Composition

\[
\text{Normalize}(p_s \parallel q_s) = \\
\llbracket V \sigma_{pc} \\
:: \llbracket_R \{ X \mapsto \text{Setzero} \mid \\
\text{Extend}(\bar{p}, \text{Shiftpcs}(\bar{q}, \text{Count}(\bar{p}))) \\
\text{Extend}(\text{Shiftpcs}(\bar{q}, \text{Count}(\bar{p})), \bar{p}) \\
\{\text{COM}(\text{alt}_p, \text{alt}_q) \mid \text{alt}_p \in \text{Alt}(\bar{p}), \text{alt}_q \in \text{Alt}(\text{Shiftpcs}(\bar{q}, \text{Count}(\bar{p}))), \\
\text{match}(\text{alt}_p, \text{alt}_q) \} \\
\} \\
\} \\
:: (u_{pc}^p \land \text{Shiftpcs}(u_{pc}^q, \text{Count}(\bar{p}))) \bowtie X \\
\rrbracket
\]

Linearization of Hybrid Chi Using Program Counters – p. 22/32
• **Count**: $\overline{P} \rightarrow \mathbb{N}$

$$\text{Count}(\overline{p}) = |\text{pcs}(\overline{p})|$$

• **Shiftpcs**: $\mathcal{D} \times \mathbb{N} \rightarrow \mathcal{D}$.  
  ($\mathcal{D}$ is a dummy structure.)  
  It shifts the subscripts of all the program counters in the given construct.

• **Alt**: $\overline{P} \rightarrow 2^{\overline{P}}$.  
  $\text{Alt}(\overline{p})$ returns all the alternatives in $\overline{p}$.

• **match**: $\overline{P} \times \overline{P} \rightarrow \text{Bool}$,

• **Extend**: $\overline{P} \times \overline{P} \rightarrow (\overline{P} \cup \{\bot\})$
\[ p = a; b \parallel c; d \]
\[ \tilde{p} = \| V \{ i_1 \mapsto \bot, i_2 \mapsto \bot \} \]
\[ :: \| R \{ X \mapsto \bar{p} \} :: (i_1 = 4 \land i_2 = 4) \bowtie X \| \]
\[
\tilde{p} = \left[ V \{ i_1 \mapsto \bot, i_2 \mapsto \bot \} \right. \\
\left. :: \left[ R \{ X \mapsto \overline{p} \} :: (i_1 = 4 \land i_2 = 4) \rightsquigarrow X \right] \right]
\]

\[
\overline{p} = \left( i_1 = 4 \right) \rightarrow a, i_1 := 2; X \\
\left( i_1 = 2 \land \neg \text{Odd}(\{i_2\}) \right) \rightarrow b, i_1 := 1; X \\
\left( i_1 = 2 \land \text{Odd}(\{i_2\}) \right) \rightarrow b, i_1 := 0, i_2 := 0 \\
\left( i_2 = 4 \right) \rightarrow c, i_2 := 2; X \\
\left( i_2 = 2 \land \neg \text{Odd}(\{i_2\}) \right) \rightarrow d, i_2 := 1; X \\
\left( i_2 = 2 \land \text{Odd}(\{i_2\}) \right) \rightarrow d, i_2 := 0, i_1 := 0
\]
A parallel composition terminates when all its components terminate.

- **Local Termination.**
  At least one other component of parallel composition has not terminated yet. Set the program counters of the terminating component to 1 instead of zero.

- **Global Termination.**
  All other components of parallel composition have terminated. Set the program counters to zero.
• Check if the values of all program counters of all the components in parallel are odd.

\[ \text{Odd}(I) = \left( \prod_{i \in I} i \mod 2 \right) = 1 \]

Odd(I) is true if all the program counters in set \( I \) have odd values.
The function \( \text{Extend} : \overline{P} \times \overline{P} \rightarrow \overline{P} \) adds proper terminating options to a parallel component. In \( \text{Normalize}(p_s \parallel q_s) \), \( \text{Extend}(\overline{p}, \overline{q}) \) does as follows:

- if \( p_s \) is a parallel composition itself then \( \text{Extend} \) adds the program counters of \( \overline{q} \) in the parity checking in terminating options of \( \overline{p} \).
- if \( p_s \) is not a parallel composition, then the function adds parity checking to its terminating options.
Observations

• Generally, we use even values of program counters to activate an alternative of the LPE. Odd values of program counters are only used in terminating options of a parallel component.

• Initially the values of program counters are maximum. The values decrease as we scan a process term. On termination the values of all program counters are set to zero.

• We reuse program counters in sequential composition, alternative composition and recursion scope etc.
• Numbers of program counters depend on the number of parallel operators in $p$.
• Termination in a parallel composition is specially handled.
Conclusion

- A tool in ASF+SDF based on the linearization algorithm will be developed.
- Some restrictions on the input form. Tail recursion, no recursion variables in parallel composition.
- No Formal Proof of the algorithm yet.