Reachability for Finite-State Process Algebras Using Static Analysis

Nataliya Skrypnyuk, Flemming Nielson
Technical University of Denmark

30. September 2011

Main idea

- Perform Static Analysis (in particular, Data Flow Analysis) on the syntax of a process algebra;
- Use the results to compute an overapproximation of the reachable states;
- If the state in question possibly reachable, construct states reachable from the initial state in one step;
- Reassess our overapproximation of reachability;
- Continue until no more states or overapproximation does not contain the state in question.

1. Process algebra with CSP synchronisation model (PA)
2. Data Flow Analysis of PA
3. Correct and complete reachability algorithm

Syntax of PA

Syntactic classes: **prefixed process variables**, **prefixed expressions**, **sums**, **recursive process definitions**, **terminal process**, **parallel compositions**, **scope restrictions**. P is a linear PA process. E is an PA process. An PA program is a uniquely labelled PA process with unique process variables.

$$
\begin{array}{lcll}
P & ::= & \mathsf{a}^{\ell}.X & | \\
 & & \mathsf{a}^{\ell}.P & | \\
 & & P + P & | \\
 & & \underline{X := P} & | \\
 & & \mathbf{0} & \\
E & ::= & P & | \\
 & & E \;{}_{||}A_{||}\; E & | \\
 & & \mathsf{hide}\ A\ \mathsf{in}\ P &
\end{array}
$$

SOS rules for PA

- *Prefixing:* $\mathsf{a}^{\ell}.P \xrightarrow[\{\ell\}]{\mathsf{a}} P$

SOS rules for PA

- *Prefixing:* $\mathsf{a}^{\ell}.P \xrightarrow[\{\ell\}]{\mathsf{a}} P$
- *Choice:* $P_1 + P_2 \xrightarrow[C]{\mathsf{a}} P_1'$ if $P_1 \xrightarrow[C]{\mathsf{a}} P_1'$

SOS rules for PA

- *Prefixing:* $\mathsf{a}^{\ell}.P \xrightarrow[\{\ell\}]{\mathsf{a}} P$
- *Choice:* $P_1 + P_2 \xrightarrow[C]{\mathsf{a}} P_1'$ if $P_1 \xrightarrow[C]{\mathsf{a}} P_1'$
- *Parallel processes:* $P_1 \,{}_{||}A_{||}\, P_2 \xrightarrow[C]{\mathsf{a}} P_1' \,{}_{||}A_{||}\, P_2$ if $P_1 \xrightarrow[C]{\mathsf{a}} P_1'$ and $\mathsf{a} \notin A$; $P_1 \,{}_{||}A_{||}\, P_2 \xrightarrow[C_1 \cup C_2]{\mathsf{a}} P_1' \,{}_{||}A_{||}\, P_2'$ if $P_1 \xrightarrow[C_1]{\mathsf{a}} P_1'$, $P_2 \xrightarrow[C_2]{\mathsf{a}} P_2'$ and $\mathsf{a} \in A$;

SOS rules for PA

- *Prefixing:* $\mathsf{a}^{\ell}.P \xrightarrow[\{\ell\}]{\mathsf{a}} P$
- *Choice:* $P_1 + P_2 \xrightarrow[C]{\mathsf{a}} P_1'$ if $P_1 \xrightarrow[C]{\mathsf{a}} P_1'$
- *Parallel processes:* $P_1 \,{}_{||}A_{||}\, P_2 \xrightarrow[C]{\mathsf{a}} P_1' \,{}_{||}A_{||}\, P_2$ if $P_1 \xrightarrow[C]{\mathsf{a}} P_1'$ and $\mathsf{a} \notin A$; $P_1 \,{}_{||}A_{||}\, P_2 \xrightarrow[C_1 \cup C_2]{\mathsf{a}} P_1' \,{}_{||}A_{||}\, P_2'$ if $P_1 \xrightarrow[C_1]{\mathsf{a}} P_1'$, $P_2 \xrightarrow[C_2]{\mathsf{a}} P_2'$ and $\mathsf{a} \in A$;
- *Internalisation:* hide A in $P \xrightarrow[C]{\tau}$ hide A in P' if $P \xrightarrow[C]{\mathsf{a}} P'$ and $\mathsf{a} \in A$; hide A in $P \xrightarrow[C]{\mathsf{a}}$ hide A in P' if $P \xrightarrow[C]{\mathsf{a}} P'$ and $\mathsf{a} \notin A$;

SOS rules for PA

- *Prefixing:* $\mathsf{a}^{\ell}.P \xrightarrow[\{\ell\}]{\mathsf{a}} P$
- *Choice:* $P_1 + P_2 \xrightarrow[C]{\mathsf{a}} P_1'$ if $P_1 \xrightarrow[C]{\mathsf{a}} P_1'$
- *Parallel processes:* $P_1 \,{}_{||}A_{||}\, P_2 \xrightarrow[C]{\mathsf{a}} P_1' \,{}_{||}A_{||}\, P_2$ if $P_1 \xrightarrow[C]{\mathsf{a}} P_1'$ and $\mathsf{a} \notin A$; $P_1 \,{}_{||}A_{||}\, P_2 \xrightarrow[C_1 \cup C_2]{\mathsf{a}} P_1' \,{}_{||}A_{||}\, P_2'$ if $P_1 \xrightarrow[C_1]{\mathsf{a}} P_1'$, $P_2 \xrightarrow[C_2]{\mathsf{a}} P_2'$ and $\mathsf{a} \in A$;
- *Internalisation:* hide A in $P \xrightarrow[C]{\tau}$ hide A in P' if $P \xrightarrow[C]{\mathsf{a}} P'$ and $\mathsf{a} \in A$; hide A in $P \xrightarrow[C]{\mathsf{a}}$ hide A in P' if $P \xrightarrow[C]{\mathsf{a}} P'$ and $\mathsf{a} \notin A$;
- *Process definition:* $\underline{X := P} \xrightarrow[C]{\mathsf{a}} P'$ if $P\left[X/\underline{X := P}\right] \xrightarrow[C]{\mathsf{a}} P'$.

Examples of PA systems

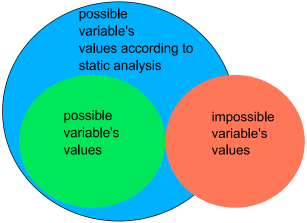
- $\underline{X := \mathsf{a}^{\ell_1}.X + \mathsf{b}^{\ell_2}.\mathbf{0}} \xrightarrow[\{\ell_1\}]{\mathsf{a}} \underline{X := \mathsf{a}^{\ell_1}.X + \mathsf{b}^{\ell_2}.\mathbf{0}}$
- $\underline{X := \mathsf{a}^{\ell_1}.X + \mathsf{b}^{\ell_2}.\mathbf{0}} \xrightarrow[\{\ell_2\}]{\mathsf{b}} \mathbf{0}$

Examples of PA systems

- $\underline{X := \mathsf{a}^{\ell_1}.X + \mathsf{b}^{\ell_2}.\mathbf{0}} \xrightarrow[\{\ell_1\}]{\mathsf{a}} \underline{X := \mathsf{a}^{\ell_1}.X + \mathsf{b}^{\ell_2}.\mathbf{0}}$
- $\underline{X := \mathsf{a}^{\ell_1}.X + \mathsf{b}^{\ell_2}.\mathbf{0}} \xrightarrow[\{\ell_2\}]{\mathsf{b}} \mathbf{0}$
- $\underline{X := \mathsf{a}^{\ell_1}.X} \parallel_{\mathsf{a}} \underline{Y := \mathsf{a}^{\ell_2}.\mathsf{b}^{\ell_3}.Y} \xrightarrow[\{\ell_1, \ell_2\}]{\mathsf{a}}$
  $\underline{X := \mathsf{a}^{\ell_1}.X} \parallel_{\mathsf{a}} \mathsf{b}^{\ell_3}.\underline{Y := \mathsf{a}^{\ell_2}.\mathsf{b}^{\ell_3}.Y}$
- $\underline{X := \mathsf{a}^{\ell_1}.X} \parallel_{\mathsf{a}} \mathsf{b}^{\ell_3}.\underline{Y := \mathsf{a}^{\ell_2}.\mathsf{b}^{\ell_3}.Y} \xrightarrow[\{\ell_3\}]{\mathsf{b}}$
  $\underline{X := \mathsf{a}^{\ell_1}.X} \parallel_{\mathsf{a}} \underline{Y := \mathsf{a}^{\ell_2}.\mathsf{b}^{\ell_3}.Y}$

Static Analysis

- Developed in the area of Program Analysis: Control Flow Analysis, Data Flow Analysis etc.
- Purpose: verifying a program by analysing program's code;



- Transferred to process calculi: verify the semantics without building full LTS, by analysing the syntax;

Data Flow Analysis of PA

- Based on **Data Flow Analysis for CCS** by H.R.Nielson and F.Nielson from 2006
- Further process calculi: BioAmbients, broadcast calculus bKlaim
- Reason: handling state space explosion
- Adjustment of the traditional Data Flow Analysis to process calculi

$$f_{state}(E) = (E \setminus kill_{state}) \cup gen_{state}$$

- Labeled Transition System states instead of program points

Transitions from E and Data Flow Analysis of E

$$E \xrightarrow[C]{\alpha} E'$$

- Transition entry: *exposed labels of E*
- Transition exit: *exposed labels* $\setminus$ *killed labels* $\cup$ *generated labels*
- Chain C corresponds to action name α
- All the labels in the chain C are exposed

Operators on PA expressions

- *Exposed* operator $\mathcal{E}$ returns labels which may "fire" in the next transition
- *Kill* operator $\mathcal{K}$ returns for a particular label those labels which must cease to be available for execution after the corresponding label has been executed
- *Generate* operator $\mathcal{G}$ returns for a particular label those labels which may become available for execution after the corresponding label has been executed
- *Chains* operator $\mathfrak{T}$ returns labels to be executed together due to synchronisation

Data Flow Analysis example

$$E \triangleq \underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Y := a^{\ell_5}.\underline{Z := d^{\ell_6}.Z}}$$

$$a \downarrow \{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$$

$$E' \triangleq b^{\ell_2}.\underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Z := d^{\ell_6}.Z}$$

Data Flow Analysis example

$$E \triangleq \underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Y := a^{\ell_5}.\underline{Z := d^{\ell_6}.Z}}$$

$$a \downarrow \{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$$

$$E' \triangleq b^{\ell_2}.\underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Z := d^{\ell_6}.Z}$$

- exposed of E: $\{\ell_1 \mapsto 1, \ell_3 \mapsto 1, \ell_5 \mapsto 1\}$

Data Flow Analysis example

$$E \triangleq \underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Y := a^{\ell_5}.\underline{Z := d^{\ell_6}.Z}}$$

$$a \downarrow \{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$$

$$E' \triangleq b^{\ell_2}.\underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Z := d^{\ell_6}.Z}$$

- exposed of E: $\{\ell_1 \mapsto 1, \ell_3 \mapsto 1, \ell_5 \mapsto 1\}$
- chains: $\{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$ etc.

Data Flow Analysis example

$$E \triangleq \underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Y := a^{\ell_5}.\underline{Z := d^{\ell_6}.Z}}$$

$$a \downarrow \{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$$

$$E' \triangleq b^{\ell_2}.\underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Z := d^{\ell_6}.Z}$$

- exposed of E: $\{\ell_1 \mapsto 1, \ell_3 \mapsto 1, \ell_5 \mapsto 1\}$
- chains: $\{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$ etc.
- $kill(\ell_1) = \{\ell_1 \mapsto 1, \ell_3 \mapsto 1\}$, $kill(\ell_5) = \{\ell_5 \mapsto 1\}$ etc.

Data Flow Analysis example

$$E \triangleq \underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Y := a^{\ell_5}.\underline{Z := d^{\ell_6}.Z}}$$

$$a \downarrow \{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$$

$$E' \triangleq b^{\ell_2}.\underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Z := d^{\ell_6}.Z}$$

- exposed of E: $\{\ell_1 \mapsto 1, \ell_3 \mapsto 1, \ell_5 \mapsto 1\}$
- chains: $\{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$ etc.
- $kill(\ell_1) = \{\ell_1 \mapsto 1, \ell_3 \mapsto 1\}$, $kill(\ell_5) = \{\ell_5 \mapsto 1\}$ etc.
- $generate(\ell_1) = \{\ell_2 \mapsto 1\}$, $generate(\ell_5) = \{\ell_6 \mapsto 1\}$ etc.

Data Flow Analysis example

$$E \triangleq \underline{X := \mathsf{a}^{\ell_1}.\mathsf{b}^{\ell_2}.X + \mathsf{c}^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Y := \mathsf{a}^{\ell_5}.\underline{Z := \mathsf{d}^{\ell_6}.Z}}$$

$$\mathsf{a} \downarrow \{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$$

$$E' \triangleq \mathsf{b}^{\ell_2}.\underline{X := \mathsf{a}^{\ell_1}.\mathsf{b}^{\ell_2}.X + \mathsf{c}^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Z := \mathsf{d}^{\ell_6}.Z}$$

- exposed of E: $\{\ell_1 \mapsto 1, \ell_3 \mapsto 1, \ell_5 \mapsto 1\}$
- chains: $\{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$ etc.
- $kill(\ell_1) = \{\ell_1 \mapsto 1, \ell_3 \mapsto 1\}$, $kill(\ell_5) = \{\ell_5 \mapsto 1\}$ etc.
- $generate(\ell_1) = \{\ell_2 \mapsto 1\}$, $generate(\ell_5) = \{\ell_6 \mapsto 1\}$ etc.
- exposed of E': $\{\ell_2 \mapsto 1, \ell_6 \mapsto 1\}$

Data Flow Analysis example

$$E \triangleq \underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Y := a^{\ell_5}.\underline{Z := d^{\ell_6}.Z}}$$

$$a \downarrow \{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$$

$$E' \triangleq b^{\ell_2}.\underline{X := a^{\ell_1}.b^{\ell_2}.X + c^{\ell_3}.\tau^{\ell_4}.X} \parallel \{a\} \parallel \underline{Z := d^{\ell_6}.Z}$$

- exposed of E: $\{\ell_1 \mapsto 1, \ell_3 \mapsto 1, \ell_5 \mapsto 1\}$
- chains: $\{\ell_1 \mapsto 1, \ell_5 \mapsto 1\}$ etc.
- $kill(\ell_1) = \{\ell_1 \mapsto 1, \ell_3 \mapsto 1\}$, $kill(\ell_5) = \{\ell_5 \mapsto 1\}$ etc.
- $generate(\ell_1) = \{\ell_2 \mapsto 1\}$, $generate(\ell_5) = \{\ell_6 \mapsto 1\}$ etc.
- exposed of E': $\{\ell_2 \mapsto 1, \ell_6 \mapsto 1\}$

$$exposed(E) \setminus kill(\ell_1) \setminus kill(\ell_5) \cup generate(\ell_1) \cup generate(\ell_5) = exposed(E')$$

Main Results for Data Flow Analysis of PA programs

- Generate, kill and chains operators on F predict all (and only) transitions from F
- Chains, generate, kill operators, chains-to-names correspondence etc. are stable under SOS transitions;
- the results of the operators on an PA program are enough to reproduce its semantics., i.e. Data Flow Analysis of PA programs not only correct, but also precise.

Theorem

Given an PA program F, then for all E such that $F \xrightarrow{*} E$, Γ and Λ mappings for labels and process definitions computed on F, we have:

- *each label is exposed in E at most once;*
- $E \xrightarrow[C]{a} E'$ *if and only if* $C \in \mathfrak{T}_\Lambda[\![F]\!]$ *and* $C \subseteq \mathcal{E}_\Gamma[\![E]\!]$;
- $\mathcal{E}_\Gamma[\![E']\!] = \mathcal{E}_\Gamma[\![E]\!] \setminus (\cup_{\ell \in C}\mathcal{K}[\![F]\!](\ell)) \bigcup (\cup_{\ell \in C}\mathcal{G}_\Gamma[\![F]\!](\ell))$.

Idea: compute overapproximation of reachable labels

- All labels exposed in the initial states are reachable;
- For other labels to be reachable there should exist a chain such that all labels in it are reachable;
- Algorithm: recursively delete from the set of reachable labels those that do not have any chain with all constituting labels in it being in the set of reachable labels

Algorithm: initialisation sep

proc $init(F)$ is
for all $\ell \in Labs(F)$ do
$\quad gchains(\ell) := \{C \in \mathfrak{T}_\Lambda[\![F]\!] | \exists \ell' \in C \text{ such that } \ell \in \mathcal{G}_\Gamma[\![F]\!](\ell')\}$
return $gchains$

Algorithm: refinement step

proc $refine(F, S, gchains)$ is
$L := S$; $gchains' := gchains$;
while $\exists \ell \in Labs(F)$ such that $(gchains'(\ell) = \emptyset) \wedge (\ell \notin S)$ do
  for all $\ell' \in Labs(F)$ do
    $gchains'(\ell') := gchains'(\ell') \setminus \{C \in \mathfrak{T}_\Lambda[\![F]\!] | \ell \in C\}$
for all $\ell \in Labs(F)$ do
  if $gchains'(\ell) \neq \emptyset$ then
    $L := L \cup \{\ell\}$;
return $L, gchains'$

Example of computing reachable labels

For $F \triangleq (\mathsf{b}^{\ell_1}.\mathsf{a}^{\ell_2}.\mathsf{c}^{\ell_3}.\mathbf{0} + \mathsf{a}^{\ell_4}.\mathsf{a}^{\ell_5}.\mathsf{d}^{\ell_6}.\mathbf{0}) \shortparallel \{\mathsf{a}, \mathsf{b}\} \shortparallel \mathsf{a}^{\ell_7}.\mathbf{0}$
we have
$init(F) = \{\ell_1 \mapsto \emptyset, \ell_2 \mapsto \emptyset, \ell_3 \mapsto \{\{\ell_2, \ell_7\}\}, \ell_4 \mapsto \emptyset, \ell_5 \mapsto \{\{\ell_4, \ell_7\}\}, \ell_6 \mapsto \{\{\ell_5, \ell_7\}\}, \ell_7 \mapsto \emptyset\}$

Example of computing reachable labels

For $F \triangleq (b^{\ell_1}.a^{\ell_2}.c^{\ell_3}.\mathbf{0} + a^{\ell_4}.a^{\ell_5}.d^{\ell_6}.\mathbf{0}) \parallel_{\{a,b\}} a^{\ell_7}.\mathbf{0}$
we have
$init(F) = \{\ell_1 \mapsto \emptyset, \ell_2 \mapsto \emptyset, \ell_3 \mapsto \{\{\ell_2, \ell_7\}\}, \ell_4 \mapsto \emptyset, \ell_5 \mapsto \{\{\ell_4, \ell_7\}\}, \ell_6 \mapsto \{\{\ell_5, \ell_7\}\}, \ell_7 \mapsto \emptyset\}$

With $(L, gchains) = refine(F, \mathcal{E}_\Gamma[\![F]\!], init(F))$,
we have
$gchains = \{\ell_1 \mapsto \emptyset, \ell_2 \mapsto \emptyset, \ell_3 \mapsto \emptyset, \ell_4 \mapsto \emptyset, \ell_5 \mapsto \{\{\ell_4, \ell_7\}\}, \ell_6 \mapsto \{\{\ell_5, \ell_7\}\}, \ell_7 \mapsto \emptyset\}$
and therefore
$L = \{\ell_1, \ell_4, \ell_5, \ell_6, \ell_7\}$.

Algorithm for reachability of $S_?$ from the initial state S_{in} of F

- Do some sanity check first: whether $S_?$ is impossible because labels in it exclude each other or whether $S_{in} = S_?$
- Add S_{in} to the Worklist
- Choose some S from the Worklist and compute overapproximation L of labels reachable from S
- If $S_? \not\subseteq L$ then break;
- Otherwise create all the transitions $S \longrightarrow S''$
- If one of S'' is equal to $S_?$ or we have encountered all S''s before then we are done
- Otherwise add all not encountered before S'' to the Worklist
- Go to p. 2

Examples

$$b^{\ell_1}.a^{\ell_2}.c^{\ell_3}.\mathbf{0} + a^{\ell_4}.a^{\ell_5}.d^{\ell_6}.\mathbf{0} \,||\{a,b\}||\, a^{\ell_7}.\mathbf{0} \xrightarrow[\{\ell_4,\ell_7\}]{a} a^{\ell_5}.d^{\ell_6}.\mathbf{0} \,||\{a,b\}||\, \mathbf{0}$$

In $(a^{\ell_1}\ldots.b^{\ell_n}.\mathbf{0} + c^{\ell'_1}\ldots.d^{\ell'_n}.\mathbf{0}) \,||\emptyset||\, e^{\ell''_1}\ldots.f^{\ell''_n}.\mathbf{0}$

the branch $c^{\ell'_1}\ldots.d^{\ell'_n}.\mathbf{0}$ interleaved with $e^{\ell''_1}\ldots.f^{\ell''_n}.\mathbf{0}$

is not explored while determining the reachability of e.g. ℓ_n

Proved results

Lemma

if $F \xrightarrow{*} E \xrightarrow{*} E'$ *for some* E *and* E' *and* L *is computed by refine on* E *then* $\mathcal{E}_\Gamma[\![E']\!] \subseteq L$

Theorem

Given a PA *program* F, *then* $F \xrightarrow{*} E$ *iff* $reach(F, \mathcal{E}_\Gamma[\![E]\!]) = true$.

Conclusions

- We have presented a complete reachability algorithm for process algebras based on Static Analysis methods
- Algorithm determines dead branches that cannot lead to the state in question
- Can be used with partial knowledge of the initial and goal states (i.e. with subsets of exposed labels)
- With efficient data structures additional overhead quadratic in the length of the systax
- Algorithm can be used just for initial state / some of the states, i.e. in the usual way Static Analysis results are used

Future work

- Other systems allowing for compositional verification – other process calculi etc.;
- Infinite semantic models (i.e., utilising Control Flow Analysis);
- Other properties checked – e.g. repeated reachability;
- Property-directed computation;
- Further reduction of the state space, e.g. through computing independent actions;
- Implementation and case studies.

Thank you for attention!
Questions?