Strategies for Optimised STG Decomposition

Mark Schaefer¹ Walter Vogler¹ Ralf Wollowski² Victor Khomenko³

¹Institute of Computer Science, University of Augsburg, Germany
²Hasso-Plattner-Institute, Potsdam, Germany
³School of Computing Science, University of Newcastle upon Tyne, UK

ACSD 2006
Overview

1. STG Decomposition
   - STGs
   - Decomposition

2. New Decomposition Strategies
   - Contraction Reordering
   - Lazy Backtracking
   - Tree Decomposition
   - Results

3. Future Research
Signal Transition Graphs are Petri nets
Transitions are labelled with signal edges
Modell for asynchronous circuits
Input signal edge activated $\rightarrow$ circuit is ready to receive it from the environment
Output/internal signal edge activated $\rightarrow$ circuit must produce this signal edge
Motivation for Decomposition

- Synthesising a circuit from an STG $N$
  - Generate the reachability graph $R$ of $N$ → state explosion
  - Derive an equation for each output signal from $R$
  - Effort more than linear in $|R|$
  - Quadratic for the naïve approach
  - Better methods work with BDDs or SAT-solving

Decomposition approach

- Split an STG into components, each producing a subset of outputs
- Perform synthesis for the components
- Advantage: Smaller reachability graphs
- Overall performance improvement

During decomposition reachability graphs must not be generated!
Motivation for Decomposition

- Synthesising a circuit from an STG $N$
  - Generate the reachability graph $R$ of $N$ → state explosion
  - Derive an equation for each output signal from $R$
  - Effort more than linear in $|R|$
  - Quadratic for the naïve approach
  - Better methods work with BDDs or SAT-solving

- Decomposition approach
  - Split an STG into components, each producing a subset of outputs
  - Perform synthesis for the components
  - Advantage: Smaller reachability graphs
  - Overall performance improvement
Motivation for Decomposition

- Synthesising a circuit from an STG $N$
  - Generate the reachability graph $R$ of $N$ → state explosion
  - Derive an equation for each output signal from $R$
  - Effort more than linear in $|R|$
  - Quadratic for the naïve approach
  - Better methods work with BDDs or SAT-solving

- Decomposition approach
  - Split an STG into components, each producing a subset of outputs
  - Perform synthesis for the components
  - Advantage: Smaller reachability graphs
  - Overall performance improvement

During decomposition reachability graphs must not be generated!
For a specification $N$, choose a partition of the output signals

For each subset produce an initial component
For a specification $N$, choose a partition of the output signals

For each subset produce an **initial component**

- Copy of $N$
- Includes outputs
- Some minimal set of additional signals as inputs
- Other signals are **lambdarised**
For a specification $N$, choose a partition of the output signals

For each subset produce an initial component
- Copy of $N$
- Includes outputs
- Some minimal set of additional signals as inputs
- Other signals are lambdarised

Reduce the components separately and non-deterministically
- Contract $\lambda$-labelled transition
- Delete redundant places and transitions
For a specification $N$, choose a partition of the output signals

For each subset produce an **initial component**
- Copy of $N$
- Includes outputs
- Some minimal set of additional signals as inputs
- Other signals are **lambdarised**

Reduce the components separately and non-deterministically
- Contract $\lambda$-labelled transition
- Delete redundant places and transitions

If necessary, **backtracking**:
- Go back to initial component
- **Delambdarise** additional signal
- Start again
Decomposition Outline

\[ N \]

\[ \lambda = N_{0} \rightarrow N_{1} \rightarrow N_{2} \rightarrow N_{3} \rightarrow \ldots \]

\[ \downarrow \]

\[ a = \text{sig}(t_{k}) \]

\[ N'_{0} \rightarrow N'_{1} \rightarrow \ldots \]

\[ \downarrow \]

\[ a' = \text{sig}(t'_{m}) \]

\[ N''_{0} \rightarrow \ldots \]
Decomposition Outline

\[ N \xrightarrow{\lambda} N_0 \]
Decomposition Outline

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \]
Decomposition Outline

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \]
Decomposition Outline

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \]
Decomposition Outline

\[
N \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \ldots
\]
Decomposition Outline

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots N_k \xrightarrow{t_k} \]

\[ a = \text{sig}(t_k) \]

\[ N' \xrightarrow{t_0'} N'_1 \xrightarrow{t_1'} N'_2 \xrightarrow{t_2'} N'_3 \xrightarrow{t_3'} \cdots N'_m \xrightarrow{t_m'} \]

\[ a' = \text{sig}(t_m') \]
Decomposition Outline

\[ \begin{align*}
N & \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots \xrightarrow{t_k} N_k \\
\downarrow a &= \text{sig}(t_k) \\
N_0' &
\end{align*} \]
$N \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots \xrightarrow{t_k} N_k$

$\downarrow \quad a = \text{sig}(t_k)$

$N_0' \xrightarrow{t_0'} N_1' \xrightarrow{t_1'} \cdots$
\[ N \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots \xrightarrow{t_k} N_k \]

\[ a = \text{sig}(t_k) \]

\[ N'_0 \xrightarrow{t'_0} N'_1 \xrightarrow{t'_1} \cdots \xrightarrow{t'_m} N'_m \]
Decomposition Outline

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots \xrightarrow{t_k} N_k \]

\[ \downarrow a = \text{sig}(t_k) \]

\[ N'_0 \xrightarrow{t'_0} N'_1 \xrightarrow{t'_1} \cdots \xrightarrow{t'_m} N'_m \]

\[ \downarrow a' = \text{sig}(t'_m) \]

\[ N''_0 \xrightarrow{t''_0} \cdots \]

\[ \vdots \]
Decomposition Outline

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots N_k \xrightarrow{t_k} \]
\[ \Downarrow a = \text{sig}(t_k) \]

\[ N'_0 \xrightarrow{t'_0} N'_1 \xrightarrow{t'_1} \cdots N'_m \xrightarrow{t'_m} \]
\[ \Downarrow a' = \text{sig}(t'_m) \]

\[ N''_0 \xrightarrow{t''_0} \cdots \]
\[ \vdots \]
\[ \cdots \cdots C \]
Transition Contraction

\[ \begin{align*}
  &Ia^- \\
  &\quad \downarrow p_1 \quad \lambda \\
  &\quad \downarrow \quad \downarrow p_3 \\
  &Ro+ \\
  &\quad \downarrow p_4 \\
  &\quad \downarrow Ia+ \\
  &\quad \downarrow \quad \downarrow \downarrow Ro+ \\
  &Ack+ \\
  &\quad \downarrow p_2 \\
  &\lambda
\end{align*} \]
Transition Contraction

\[
\begin{align*}
&\text{Ia}^- \quad \text{Ack}^+ \\
p_1 \quad p_2 \\
&\text{Ro}^+ \\
p_3 \quad p_4 \\
&\text{Ia}^+ \quad \text{Ro}^+
\end{align*}
\]

\[
\begin{align*}
&\text{Ia}^- \\
p_1 p_4 \quad p_2 p_3 \\
&\text{Ro}^+ \\
p_1 p_3 \quad p_2 p_4 \\
&\text{Ia}^+ \\
&\text{Ro}^+
\end{align*}
\]
Backtracking is performed if no more $\lambda$-transition can be contracted, because the contraction
- ... is not defined (loops, arcweight $\geq 2$)
- ... is not-secure (language changed)
- ... generates structural **auto-conflict**
Auto-Conflicts

Structural auto-conflict
Dynamic auto-conflict
New Decomposition Strategies

- Contraction reordering
- Lazy backtracking
- Tree decomposition
Contraction Reordering

- Contraction of ‘good’ transitions produces few new places

- Contraction of ‘bad’ transitions produces many new places

- Observation: Contracting ‘good’ transitions first, results in smaller intermediate STGs (important for looking for redundant places)

- Sometimes, contracting ‘bad’ transitions first results in bigger final STGs

- Therefore, contract ‘good’ transitions first
Contraction Reordering

- Contraction of ‘good’ transitions produces few new places

\[ \lambda \]

Observation: Contracting ‘good’ transitions first results in smaller intermediate STGs (important for looking for redundant places).

Sometimes, contracting ‘bad’ transitions first results in bigger final STGs.

Therefore, contract ‘good’ transitions first.
Contraction Reordering

- Contraction of ‘good’ transitions produces few new places
- Contraction of ‘bad’ transitions produces many new places
Contraction Reordering

- Contraction of ‘good’ transitions produces few new places
- Contraction of ‘bad’ transitions produces many new places
- Observation: Contracting ‘good’ transitions first, results in smaller intermediate STGs (important for looking for redundant places)

Sometimes, contracting ‘bad’ transitions first results in bigger final STGs

Therefore, contract ‘good’ transitions first

Mark Schaefer (University of Augsburg)
Contraction Reordering

- Contraction of ‘good’ transitions produces few new places
- Contraction of ‘bad’ transitions produces many new places
- Observation: Contracting ‘good’ transitions first, results in smaller intermediate STGs (important for looking for redundant places)
- Sometimes, contracting ‘bad’ transitions first results in bigger final STGs
- Therefore, contract ‘good’ transitions first
Lazy Backtracking

- Contract transitions grouped by former signals
- After a signal was completely contracted save the intermediate result
- When backtracking, don’t start at the beginning
Lazy Backtracking

- Contract transitions grouped by former signals
- After a signal was completely contracted save the intermediate result
- When backtracking, don’t start at the beginning

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{a_0} \cdots \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j} N'_k \]

Mark Schaefer (University of Augsburg)  STG Decomposition Strategies  ACSD 2006  13 / 18
Lazy Backtracking

- Contract transitions grouped by former signals
- After a signal was completely contracted save the intermediate result
- When backtracking, don’t start at the beginning

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{a_0} \cdots \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j} N_j' \]
Lazy Backtracking

- Contract transitions grouped by former signals
- After a signal was completely contracted save the intermediate result
- When backtracking, don’t start at the beginning

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{a_0} \cdots \]

\[ \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j} \]

\[ \downarrow a_j \quad \downarrow a_j \]

\[ N'_{j-1} \quad N'_{j} \]
Lazy Backtracking

- Contract transitions grouped by former signals
- After a signal was completely contracted save the intermediate result
- When backtracking, don’t start at the beginning

$$N \xrightarrow{\lambda} N_0 \xrightarrow{a_0} \cdots N_{k-1} \xrightarrow{a_{k-1}} N_k \xrightarrow{a_k} \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j}$$
Lazy Backtracking

- Contract transitions grouped by former signals
- After a signal was completely contracted save the intermediate result
- When backtracking, don’t start at the beginning

\[
N \xrightarrow{\lambda} N_0 \xrightarrow{a_0} \cdots N_{k-1} \xrightarrow{a_{k-1}} N_k \xrightarrow{a_k} \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j} \nabla \xrightarrow{a_j} N'_{k-1} \xrightarrow{a_k} \nabla \xrightarrow{a_j} N'_{j-1} \xrightarrow{a_j} N'_j \xrightarrow{a_j} N''_j
\]
Lazy Backtracking

- Contract transitions grouped by former signals
- After a signal was completely contracted save the intermediate result
- When backtracking, don’t start at the beginning

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{a_0} \cdots N_{k-1} \xrightarrow{a_{k-1}} N_k \xrightarrow{a_k} \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j} \]

\[ \downarrow a_j \quad \downarrow a_j \quad \downarrow a_j \]

\[ N'_k \quad N'_{j-1} \quad N'_j \]

\[ \downarrow a_k, a_j \quad \downarrow a_k \]

\[ N'_{k-1} \quad N''_k \]
Lazy Backtracking

- Contract transitions grouped by former signals
- After a signal was completely contracted save the intermediate result
- When backtracking, don’t start at the beginning

\[
N \xrightarrow{\lambda} N_0 \xrightarrow{a_0} \cdots \xrightarrow{a_{k-1}} N_k \xrightarrow{a_k} \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j}
\]

\[
\downarrow a_j \quad \downarrow a_j \quad \downarrow a_j
\]

\[
N'_k \quad N'_{j-1} \quad N'_j
\]

\[
\downarrow a_k, a_j \quad \downarrow a_k
\]

\[
N'_{k-1} \quad N''_{k-1}
\]
Lazy Backtracking

- Contract transitions grouped by former signals
- After a signal was completely contracted save the intermediate result
- When backtracking, don’t start at the beginning

\[ N \xrightarrow{\lambda} N_0 \xrightarrow{a_0} \cdots N_{k-1} \xrightarrow{a_{k-1}} N_k \xrightarrow{a_k} \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j} \]

\[ \Downarrow a_j \quad \Downarrow a_j \quad \Downarrow a_j \]

\[ N'_k \quad N'_{j-1} \quad N'_j \]

\[ \Downarrow a_k, a_j \quad \Downarrow a_k \]

\[ N'_{k-1} \quad N''_k \]

\[ \Downarrow a_{k-1} \]

\[ N''_{k-1} \xrightarrow{\cdots} \]
Components are generated separately even if they are similar (nearly the same signals should be contracted)

Tree decomposition:
- Group contractions by former signals (as for lazy backtracking)
- Find an appropriate order of contractions
- Reuse intermediate results
Tree Decomposition
Tree Decomposition
Tree Decomposition
Tree Decomposition
Tree Decomposition
Tree Decomposition
Tree Decomposition
## Results

(More results and detailed discussion in the paper)

<table>
<thead>
<tr>
<th>STG</th>
<th>Tree</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t (sec)</td>
<td>size</td>
</tr>
<tr>
<td>2pp.arb.nch.9.csc</td>
<td>2</td>
<td>227</td>
</tr>
<tr>
<td>2pp.arb.nch.9</td>
<td>2</td>
<td>198</td>
</tr>
<tr>
<td>2pp.arb.nch.12.csc</td>
<td>6</td>
<td>275</td>
</tr>
<tr>
<td>3pp.arb.nch.9</td>
<td>4</td>
<td>350</td>
</tr>
<tr>
<td>3pp.arb.nch.12.csc e</td>
<td>14</td>
<td>537</td>
</tr>
<tr>
<td>3pp.arb.nch.12</td>
<td>15</td>
<td>422</td>
</tr>
</tbody>
</table>
Results

- In most cases all new strategies perform much better than basic decomposition
- In most cases the components get smaller for every strategy
- Especially tree decomposition reduces runtimes while producing small components
Future Research

Decomposition is fast enough now $\rightarrow$ improve the *quality* of the results

- Combine tree decomposition with CSC solving
- Combine decomposition with Handshakecircuits, e.g. generated by Balsa