Strategies for Optimised STG Decomposition

Mark Schaefer[1] Walter Vogler[1] Ralf Wollowski[2] Victor Khomenko[3]

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

ACSD 2006

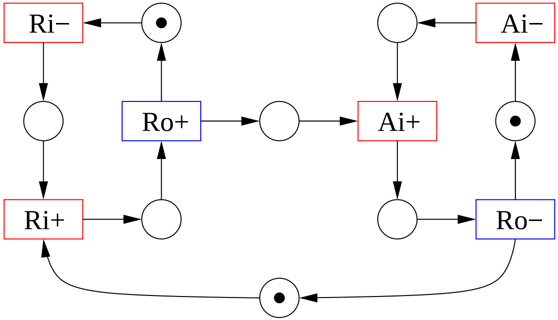
Overview

1. STG Decomposition
   - STGs
   - Decomposition
2. New Decomposition Strategies
   - Contraction Reordering
   - Lazy Backtracking
   - Tree Decomposition
   - Results
3. Future Research

STGs

- 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

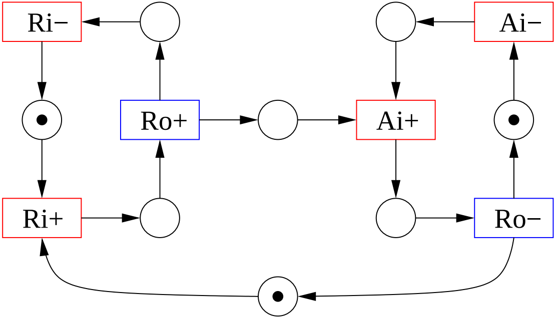
STGs



inputs

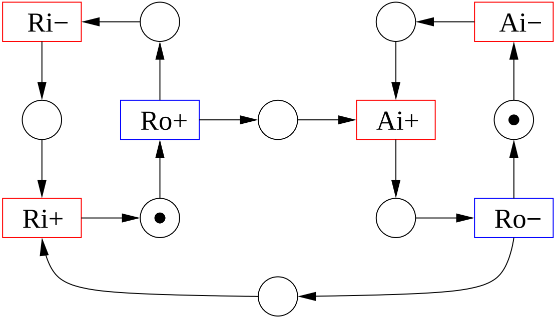
outputs

STGs



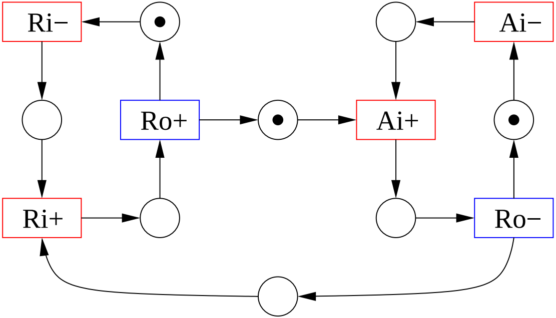
inputs outputs

STGs



inputs outputs

STGs



inputs

outputs

Motivation for Decomposition

- Synthesising a ciruit from an STG N
  - Generate the reachability graph R of $N \longrightarrow$ 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

Motivation for Decomposition

- Synthesising a ciruit from an STG N
  - Generate the reachability graph R of $N \longrightarrow$ 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 ciruit from an STG N
  - Generate the reachability graph R of $N \longrightarrow$ 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!

Decomposition Outline

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

Decomposition Outline

- 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**

Decomposition Outline

- 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 seperately and non-deterministicly
  - Contract λ-labelled transition
  - Delete redundant places and transitions

Decomposition Outline

- 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 seperately and non-deterministicly
  - Contract λ-labelled transition
  - Delete redundant places and transitions
- If neccessary, **backtracking**:
  - Go back to initial component
  - **Delambdarise** additional signal
  - Start again

Decomposition Outline

N

Decomposition Outline

$N \stackrel{\lambda}{\Longrightarrow} N_0$

Decomposition Outline

$$N \stackrel{\lambda}{\Longrightarrow} N_0 \xrightarrow{t_0} N_1$$

Decomposition Outline

$$N \xRightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2$$

Decomposition Outline

$$N \overset{\lambda}{\Longrightarrow} N_0 \overset{t_0}{\longrightarrow} N_1 \overset{t_1}{\longrightarrow} N_2 \overset{t_2}{\longrightarrow} N_3$$

Decomposition Outline

$$N \stackrel{\lambda}{\Longrightarrow} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots$$

Decomposition Outline

$$N \stackrel{\lambda}{\Longrightarrow} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots N_k \xrightarrow{t_k}$$

Decomposition Outline

$$N \stackrel{\lambda}{\Longrightarrow} N_0 \stackrel{t_0}{\longrightarrow} N_1 \stackrel{t_1}{\longrightarrow} N_2 \stackrel{t_2}{\longrightarrow} N_3 \stackrel{t_3}{\longrightarrow} \cdots \; N_k \stackrel{t_k}{\longrightarrow}$$

$$\Downarrow a = sig(t_k)$$

$$N_0'$$

Decomposition Outline

$$N \stackrel{\lambda}{\Longrightarrow} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots \; N_k \xrightarrow{t_k}$$

$$\Downarrow a = sig(t_k)$$

$$N'_0 \xrightarrow{t'_0} N'_1 \xrightarrow{t'_1} \cdots$$

Decomposition Outline

$$N \xRightarrow{\lambda} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots \; N_k \xrightarrow{t_k}$$

$$\Downarrow a = sig(t_k)$$

$$N'_0 \xrightarrow{t'_0} N'_1 \xrightarrow{t'_1} \cdots \; N'_m \xrightarrow{t'_m}$$

Decomposition Outline

$$N \overset{\lambda}{\Longrightarrow} N_0 \xrightarrow{t_0} N_1 \xrightarrow{t_1} N_2 \xrightarrow{t_2} N_3 \xrightarrow{t_3} \cdots \; N_k \xrightarrow{t_k}$$

$$\Downarrow a = sig(t_k)$$

$$N'_0 \xrightarrow{t'_0} N'_1 \xrightarrow{t'_1} \cdots \; N'_m \xrightarrow{t'_m}$$

$$\Downarrow a' = sig(t'_m)$$

$$N''_0 \xrightarrow{t''_0} \cdots$$

$$\vdots$$

Decomposition Outline

$$N \stackrel{\lambda}{\Longrightarrow} N_0 \stackrel{t_0}{\longrightarrow} N_1 \stackrel{t_1}{\longrightarrow} N_2 \stackrel{t_2}{\longrightarrow} N_3 \stackrel{t_3}{\longrightarrow} \cdots \; N_k \stackrel{t_k}{\longrightarrow}$$

$$\Downarrow a = sig(t_k)$$

$$N'_0 \stackrel{t'_0}{\longrightarrow} N'_1 \stackrel{t'_1}{\longrightarrow} \cdots \; N'_m \stackrel{t'_m}{\longrightarrow}$$

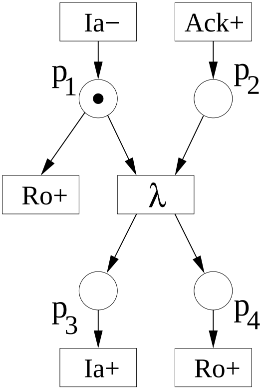
$$\Downarrow a' = sig(t'_m)$$

$$N''_0 \stackrel{t''_0}{\longrightarrow} \cdots$$

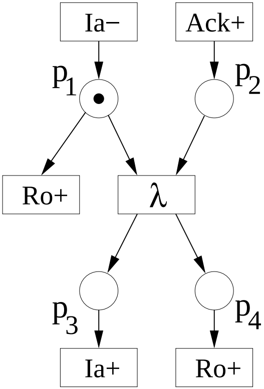
$$\vdots$$

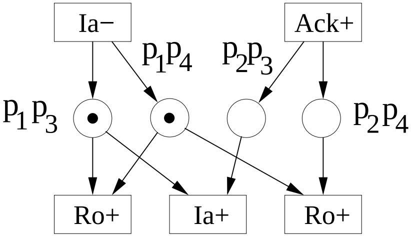
$$\cdots \qquad \cdots C$$

Transition Contraction



Transition Contraction



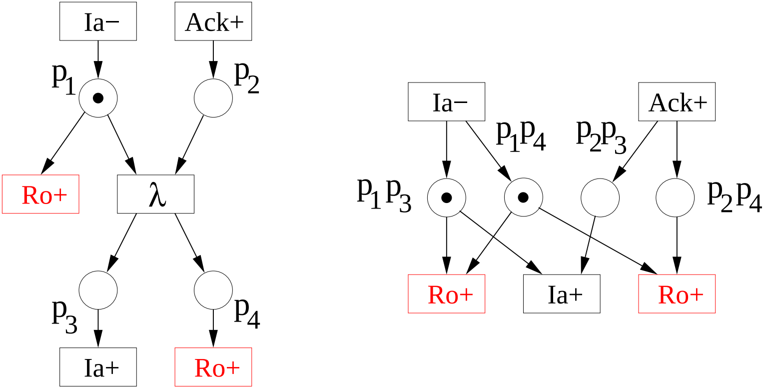


Backtracking

Backtracking is performed if no more λ-transition can be contracted, because the contraction

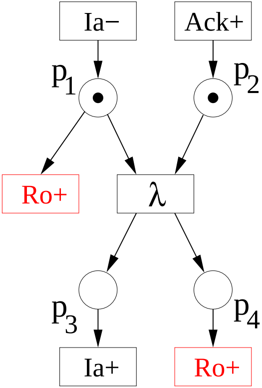
- . . . is not defined (loops, arcweight $\geq$2)
- . . . is not-secure (langugage changed)
- . . . generates structural **auto-conflict**

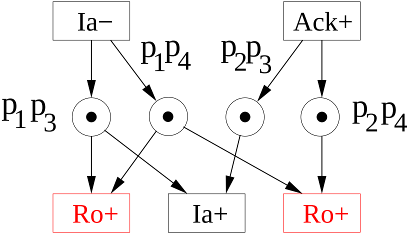
Auto-Conflicts



Structural auto-conflict

Auto-Conflicts





Dynamic auto-conflict

New Decomposition Strategies

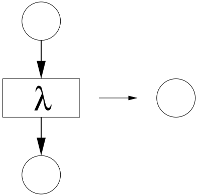
- Contraction reordering
- Lazy backtracking
- Tree decomposition

Contraction Reordering

- Contraction of 'good' transitions produces few new places

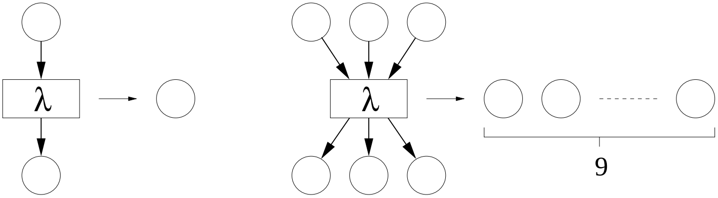
Contraction Reordering

- Contraction of 'good' transitions produces few new places



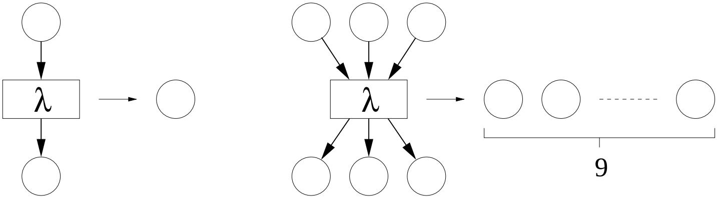
Contraction Reordering

- Contraction of 'good' transitions produces few new places
- Contraction of 'bad' transitions produces many new places



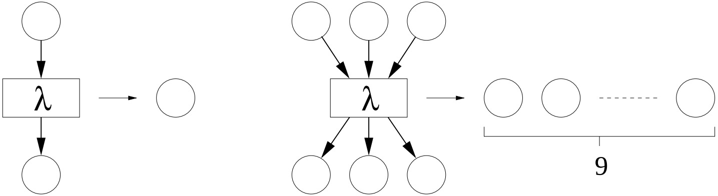
Contraction Reordering

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



Contraction Reordering

- Contraction of ‘good’ transitions produces few new places
- Contraction of ‘bad’ transitions produces many new places
- Observation: Contracting ‘good’ transtions 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

$N \stackrel{\lambda}{\Longrightarrow} N_0 \xrightarrow{a_0} \cdots$ $\cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j}$

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

$$N \stackrel{\lambda}{\Longrightarrow} N_0 \xrightarrow{a_0} \cdots \qquad \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j}$$

$$N_j \Downarrow a_j \; N'_j$$

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

$$N \overset{\lambda}{\Longrightarrow} N_0 \xrightarrow{a_0} \cdots \qquad \cdots N_{j-1} \xrightarrow{a_{j-1}} N_j \xrightarrow{a_j}$$

$$N_{j-1} \Downarrow a_j \; N'_{j-1} \qquad N_j \Downarrow a_j \; N'_j$$

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

$$
\begin{array}{ccccccccccccc}
N & \overset{\lambda}{\Longrightarrow} & N_0 & \xrightarrow{a_0} & \cdots & N_{k-1} & \xrightarrow{a_{k-1}} & N_k & \xrightarrow{a_k} & \cdots N_{j-1} & \xrightarrow{a_{j-1}} & {\color{red}N_j} & {\color{red}\xrightarrow{a_j}} \\
 & & & & & & & \Downarrow a_j & & \Downarrow a_j & & \Downarrow a_j & \\
 & & & & & & & N'_k & & {\color{red}N'_{j-1}} & & {\color{red}N'_j} &
\end{array}
$$

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

$$
\begin{array}{ccccccccccc}
N \stackrel{\lambda}{\Longrightarrow} N_0 & \stackrel{a_0}{\longrightarrow} & \cdots & N_{k-1} & \stackrel{a_{k-1}}{\longrightarrow} & N_k & \stackrel{a_k}{\longrightarrow} & \cdots N_{j-1} & \stackrel{a_{j-1}}{\longrightarrow} & N_j & \stackrel{a_j}{\longrightarrow} \\
 & & & & & \Downarrow a_j & & \Downarrow a_j & & \Downarrow a_j & \\
 & & & & & N'_k & & N'_{j-1} & & N'_j & \\
 & & & & & \Downarrow a_k & & & & & \\
 & & & & & N''_k & & & & &
\end{array}
$$

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

$$
\begin{array}{ccccccccccc}
N & \stackrel{\lambda}{\Longrightarrow} & N_0 & \xrightarrow{a_0} & \cdots & N_{k-1} & \xrightarrow{a_{k-1}} & N_k & \xrightarrow{a_k} \cdots & N_{j-1} & \xrightarrow{a_{j-1}} & N_j & \xrightarrow{a_j} \\
 & & & & & & & \Downarrow a_j & & \Downarrow a_j & & \Downarrow a_j & \\
 & & & & & & & N'_k & & N'_{j-1} & & N'_j & \\
 & & & & & \Downarrow a_k, a_j & & \Downarrow a_k & & & & & \\
 & & & & & N'_{k-1} & & N''_k & & & & &
\end{array}
$$

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

$$
\begin{array}{ccccccccccc}
N \stackrel{\lambda}{\Longrightarrow} N_0 \xrightarrow{a_0} \cdots & N_{k-1} & \xrightarrow{a_{k-1}} & N_k & \xrightarrow{a_k} \cdots & N_{j-1} & \xrightarrow{a_{j-1}} & N_j & \xrightarrow{a_j} \\
 & & & \Downarrow a_j & & \Downarrow a_j & & \Downarrow a_j & \\
 & & & N'_k & & N'_{j-1} & & N'_j & \\
 & \Downarrow a_k, a_j & & \Downarrow a_k & & & & & \\
 & N'_{k-1} & & N''_k & & & & & \\
 & \Downarrow a_{k-1} & & & & & & & \\
 & N''_{k-1} & & & & & & &
\end{array}
$$

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

$$
\begin{array}{lllllllll}
N \stackrel{\lambda}{\Longrightarrow} N_0 \xrightarrow{a_0} \cdots & N_{k-1} & \xrightarrow{a_{k-1}} & N_k & \xrightarrow{a_k} \cdots & N_{j-1} & \xrightarrow{a_{j-1}} & N_j & \xrightarrow{a_j} \\
 & & & \Downarrow a_j & & \Downarrow a_j & & \Downarrow a_j & \\
 & & & N'_k & & N'_{j-1} & & N'_j & \\
 & \Downarrow a_k, a_j & & \Downarrow a_k & & & & & \\
 & N'_{k-1} & & N''_k & & & & & \\
 & \Downarrow a_{k-1} & & & & & & & \\
 & N''_{k-1} & \longrightarrow \cdots & & & & & &
\end{array}
$$

Tree Decomposition

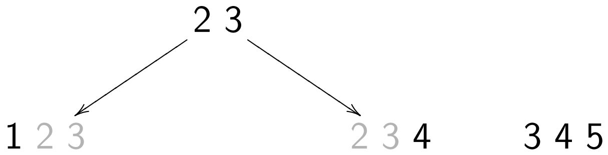
- Components are generated separately even if they are similar (nearly the same signals should be contracted)
- Tree decomposition:
  - Group contractions by former signals (as for lazy backtracking)
  - Find an appropriate order of contractions
  - Reuse intermediate results

Tree Decomposition

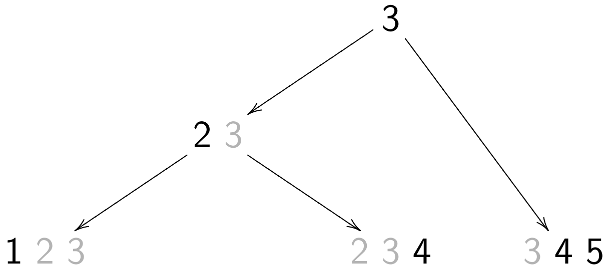
Tree Decomposition

1 2 3 2 3 4 3 4 5

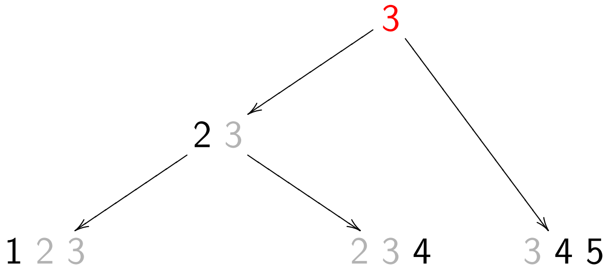
Tree Decomposition



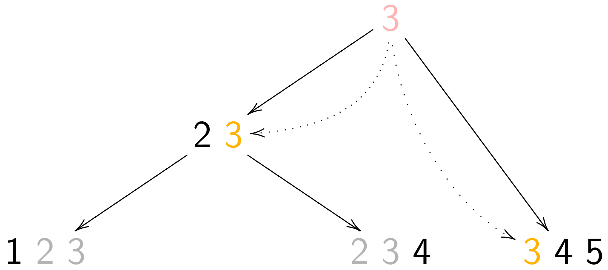
Tree Decomposition



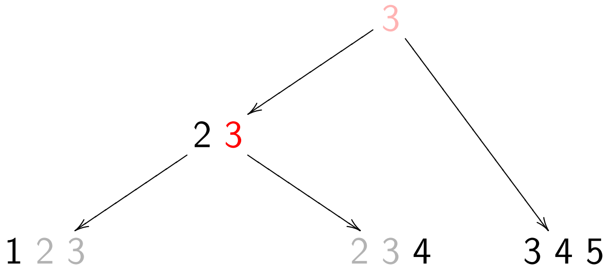
Tree Decomposition

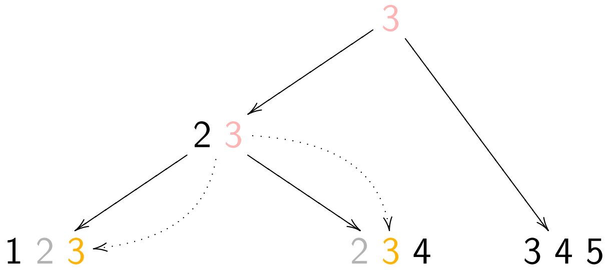


Tree Decomposition

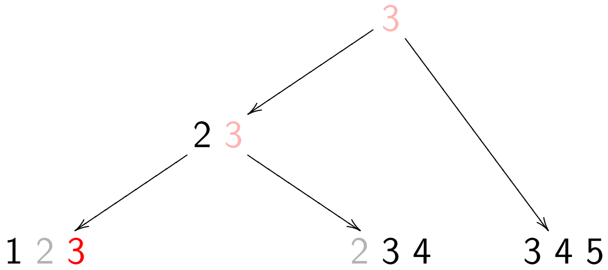


Tree Decomposition

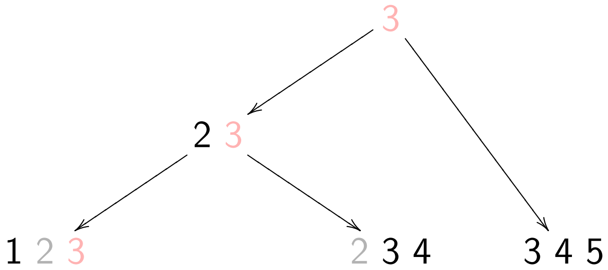



3
2 3
1 2 3
2 3 4
3 4 5

Tree Decomposition



Tree Decomposition



Results

	tree		random	
STG	t (sec)	size	t (sec)	size
2pp.arb.nch.9.csc	2	227	58	4939
2pp.arb.nch.9	2	198	71	2398
2pp.arb.nch.12.csc	6	275	158	3083
3pp.arb.nch.9	4	350	281	13198
3pp.arb.nch.12.csc e	14	537	627	7289
3pp.arb.nch.12	15	422	632	7142

(More results and detailed discussion in the paper)

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 $\longrightarrow$ improve the *quality* of the results

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