Tree-like automata synchronisation topologies and their reductions

Michał Knapik, Laure Petrucci

TDCS seminar, 16th July 2020

ICS PAS/LIPN, CNRS

Outline

Motivations and Contributions

Asynchronous Products, Synchronisation Topologies

Tree Topologies, Reductions

Motivations and Contributions

Motivations

- Our typical pipeline of system model design and verification:
  - Abstract components $\{\mathcal{M}_i\}_{i=1}^{n}$ and interfaces.
  - Specify relevant property φ.
  - Verify ϕ over (huge) model: $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n \models \varphi$.

Motivations

- Our typical pipeline of system model design and verification:
  - Abstract components $\{\mathcal{M}_i\}_{i=1}^n$ and interfaces.
  - Specify relevant property φ.
  - Verify ϕ over (huge) model: $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n \models \varphi$.
- The usual bottleneck: computing $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n$.

Motivations

- Our typical pipeline of system model design and verification:
  - Abstract components $\{\mathcal{M}_i\}_{i=1}^n$ and interfaces.
  - Specify relevant property φ.
  - Verify ϕ over (huge) model: $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n \models \varphi$.
- The usual bottleneck: computing $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n$.
- Assume-guarantee approach can sometimes alleviate this,
  e.g. find a model $\mathcal{A}$ that subsumes $\mathcal{M}_1$ and check $\mathcal{A} \| \ldots \| \mathcal{M}_n \models \varphi$.

Motivations

- Our typical pipeline of system model design and verification:
  - Abstract components $\{\mathcal{M}_i\}_{i=1}^n$ and interfaces.
  - Specify relevant property φ.
  - Verify ϕ over (huge) model: $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n \models \varphi$.
- The usual bottleneck: computing $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n$.
- Assume-guarantee approach can sometimes alleviate this, e.g. find a model $\mathcal{A}$ that subsumes $\mathcal{M}_1$ and check $\mathcal{A} \| \ldots \| \mathcal{M}_n \models \varphi$. Might work for some properties and models, but still needs costly computation of large parallel product.

Motivations

- Our typical pipeline of system model design and verification:
  - Abstract components $\{\mathcal{M}_i\}_{i=1}^n$ and interfaces.
  - Specify relevant property φ.
  - Verify ϕ over (huge) model: $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n \models \varphi$.
- The usual bottleneck: computing $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n$.
- Assume-guarantee approach can sometimes alleviate this, e.g. find a model $\mathcal{A}$ that subsumes $\mathcal{M}_1$ and check $\mathcal{A} \| \ldots \| \mathcal{M}_n \models \varphi$. Might work for some properties and models, but still needs costly computation of large parallel product.
- So it'd be good to have tools for static analysis and transformation of network $\{\mathcal{M}_i\}_{i=1}^n$ *before* computing parallel product.

Contributions (current / expected)

Current

- (Trivial) definition of synchronisation topology.
- A technique for reachability-preserving reduction of tree-like topologies for a class of *very simple automata*. The result is polynomial w.r.t. number of components.
- *The technique does not preserve safety properties - example.*
- Implementation and initial tests.

Expected (in the final version of this work)

- Generalisation of reachability-preserving reductions for tree-like topologies for any type of automata + implementation.

Expected (future work)

- Lot of things: different topologies, properties, applications, etc.

Asynchronous Products, Synchronisation Topologies

Labeled Transition Systems

Labelled Transition System ($\mathcal{LTS}$)

$\mathcal{LTS}$ is tuple $\mathcal{M} = \langle \mathcal{S}, s^0, \mathit{Acts}, \rightarrow, \mathcal{L} \rangle$ where:

1. $\mathcal{S}$ is a finite set of states and $s^0 \in \mathcal{S}$ the initial state;
2. Acts is a finite set of action names;
3. $\rightarrow \subseteq \mathcal{S} \times \mathit{Acts} \times \mathcal{S}$ is a transition relation;
4. $\mathcal{L} \colon \mathcal{S} \rightarrow 2^{\mathcal{PV}}$ labels states with propositions.

(We put $\mathit{acts}(\mathcal{M}) = \mathit{Acts}$, $\mathit{states}(\mathcal{M}) = \mathcal{S}$, etc.)

Asynchronous Product

Asynchronous Product ($\mathcal{M}_1||\mathcal{M}_2$)

- $\mathcal{M}_i = \langle \mathcal{S}_i, s_i^0, \rightarrow_i, Acts_i \rangle$: $\mathcal{LTS}$, for $i \in \{1, 2\}$
- $\mathcal{M}_1||\mathcal{M}_2 = \langle \mathcal{S}_1 \times \mathcal{S}_2, (s_1^0, s_2^0), \rightarrow, Acts_1 \cup Acts_2 \rangle$ with trans. rules:

$$\frac{act \in Acts_1 \setminus Acts_2 \wedge s_1 \xrightarrow{act}_1 s_1'}{(s_1, s_2) \xrightarrow{act} u(s_1', s_2)} \quad \text{(non-sync left trans.)}$$

$$\frac{act \in Acts_2 \setminus Acts_1 \wedge s_2 \xrightarrow{act}_2 s_2'}{(s_1, s_2) \xrightarrow{act} (s_1, s_2')} \quad \text{(non-sync right trans.)}$$

$$\frac{act \in Acts_1 \cap Acts_2 \wedge s_1 \xrightarrow{act}_1 s_1' \wedge s_2 \xrightarrow{act}_2 s_2'}{(s_1, s_2) \xrightarrow{act} (s_1', s_2')} \quad \text{(sync. trans.)}$$

- (Generalised to any number of components in the usual way.)

Asynchronous Product

Asynchronous Product ($\mathcal{M}_1||\mathcal{M}_2$)

- $\mathcal{M}_i = \langle \mathcal{S}_i, s_i^0, \rightarrow_i, Acts_i \rangle$: $\mathcal{LTS}$, for $i \in \{1, 2\}$
- $\mathcal{M}_1||\mathcal{M}_2 = \langle \mathcal{S}_1 \times \mathcal{S}_2, (s_1^0, s_2^0), \rightarrow, Acts_1 \cup Acts_2 \rangle$ with trans. rules:

$$\frac{act \in Acts_1 \setminus Acts_2 \wedge s_1 \xrightarrow{act}_1 s_1'}{(s_1, s_2) \xrightarrow{act} u(s_1', s_2)} \quad \text{(non-sync left trans.)}$$

$$\frac{act \in Acts_2 \setminus Acts_1 \wedge s_2 \xrightarrow{act}_2 s_2'}{(s_1, s_2) \xrightarrow{act} (s_1, s_2')} \quad \text{(non-sync right trans.)}$$

$$\frac{act \in Acts_1 \cap Acts_2 \wedge s_1 \xrightarrow{act}_1 s_1' \wedge s_2 \xrightarrow{act}_2 s_2'}{(s_1, s_2) \xrightarrow{act} (s_1', s_2')} \quad \text{(sync. trans.)}$$

- (Generalised to any number of components in the usual way.)

Asynchronous Product

Asynchronous Product ($\mathcal{M}_1||\mathcal{M}_2$)

- $\mathcal{M}_i = \langle \mathcal{S}_i, s_i^0, \rightarrow_i, Acts_i \rangle$: $\mathcal{LTS}$, for $i \in \{1, 2\}$
- $\mathcal{M}_1||\mathcal{M}_2 = \langle \mathcal{S}_1 \times \mathcal{S}_2, (s_1^0, s_2^0), \rightarrow, Acts_1 \cup Acts_2 \rangle$ with trans. rules:

$$\frac{act \in Acts_1 \setminus Acts_2 \wedge s_1 \xrightarrow{act}_1 s_1'}{(s_1, s_2) \xrightarrow{act} u(s_1', s_2)} \quad \text{(non-sync left trans.)}$$

$$\frac{act \in Acts_2 \setminus Acts_1 \wedge s_2 \xrightarrow{act}_2 s_2'}{(s_1, s_2) \xrightarrow{act} (s_1, s_2')} \quad \text{(non-sync right trans.)}$$

$$\frac{act \in Acts_1 \cap Acts_2 \wedge s_1 \xrightarrow{act}_1 s_1' \wedge s_2 \xrightarrow{act}_2 s_2'}{(s_1, s_2) \xrightarrow{act} (s_1', s_2')} \quad \text{(sync. trans.)}$$

- (Generalised to any number of components in the usual way.)

Asynchronous Product

Asynchronous Product ($\mathcal{M}_1||\mathcal{M}_2$)

- $\mathcal{M}_i = \langle \mathcal{S}_i, s_i^0, \rightarrow_i, Acts_i \rangle$: $\mathcal{LTS}$, for $i \in \{1, 2\}$
- $\mathcal{M}_1||\mathcal{M}_2 = \langle \mathcal{S}_1 \times \mathcal{S}_2, (s_1^0, s_2^0), \rightarrow, Acts_1 \cup Acts_2 \rangle$ with trans. rules:

$$\frac{act \in Acts_1 \setminus Acts_2 \wedge s_1 \xrightarrow{act}_1 s_1'}{(s_1, s_2) \xrightarrow{act} u(s_1', s_2)} \quad \text{(non-sync left trans.)}$$

$$\frac{act \in Acts_2 \setminus Acts_1 \wedge s_2 \xrightarrow{act}_2 s_2'}{(s_1, s_2) \xrightarrow{act} (s_1, s_2')} \quad \text{(non-sync right trans.)}$$

$$\frac{act \in Acts_1 \cap Acts_2 \wedge s_1 \xrightarrow{act}_1 s_1' \wedge s_2 \xrightarrow{act}_2 s_2'}{(s_1, s_2) \xrightarrow{act} (s_1', s_2')} \quad \text{(sync. trans.)}$$

- (Generalised to any number of components in the usual way.)

Synchronisation Topology

Synchronisation Topology $\mathcal{G}$

- $Net = \{\mathcal{M}_i\}_{i=1}^{n}$: $\mathcal{LTS}$, for $i \in \{1, \ldots, n\}$
- $\mathcal{G} = \langle Net, \mathcal{T} \rangle$: a graph with vertices Net and edges $\mathcal{T}$ s.t. $(\mathcal{M}_i, \mathcal{M}_j) \in \mathcal{T}$ iff $i \neq j$ and $acts(\mathcal{M}_i) \cap acts(\mathcal{M}_j) \neq \emptyset$

Synchronisation Topology

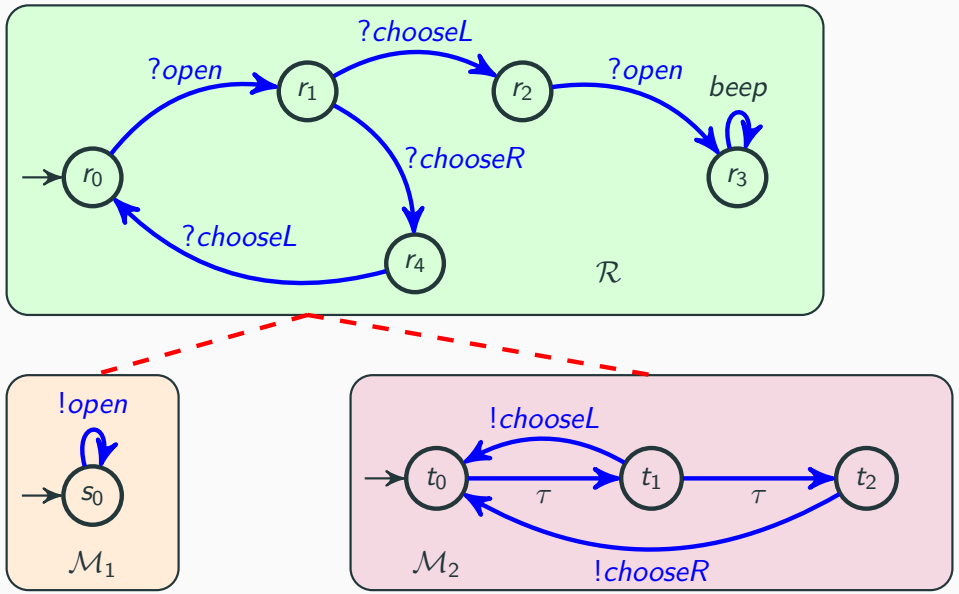
Synchronisation Topology $\mathcal{G}$

- $Net = \{\mathcal{M}_i\}_{i=1}^{n}$: $\mathcal{LTS}$, for $i \in \{1, \ldots, n\}$
- $\mathcal{G} = \langle Net, \mathcal{T} \rangle$: a graph with vertices Net and edges $\mathcal{T}$ s.t. $(\mathcal{M}_i, \mathcal{M}_j) \in \mathcal{T}$ iff $i \neq j$ and $acts(\mathcal{M}_i) \cap acts(\mathcal{M}_j) \neq \emptyset$

Tree Topologies, Reductions

Tree Topologies and Live-reset Automata

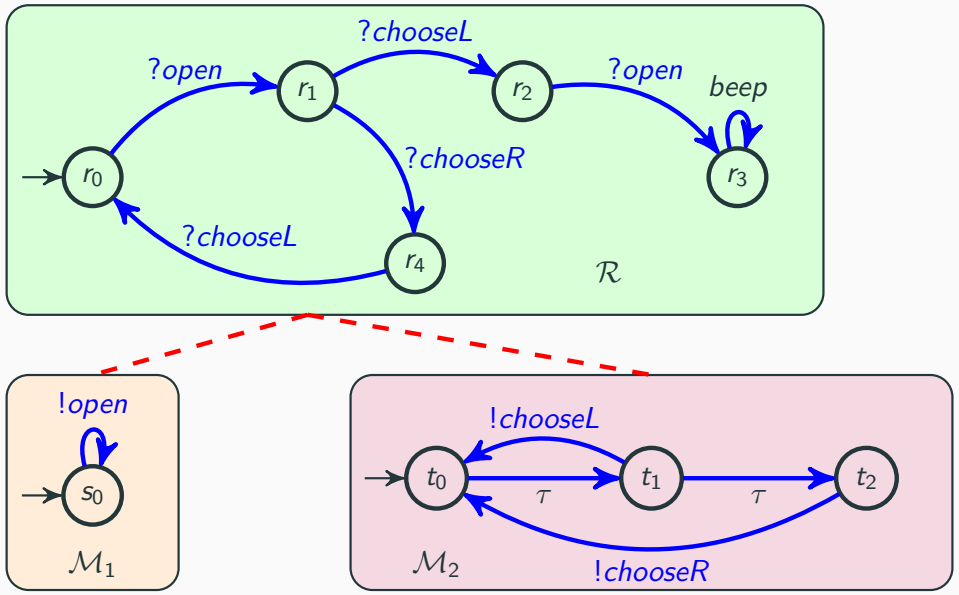
First restriction: $\mathcal{G}$ is laid out as a tree



- $\mathit{parent}(\mathcal{M}_1) = \mathit{parent}(\mathcal{M}_2) = \mathcal{R}$, $\mathit{children}(\mathcal{R}) = \{\mathcal{M}_1, \mathcal{M}_2\}$
- $\mathit{downacts}(\mathcal{M})$: actions over which $\mathcal{M}$ synchronises with any child, e.g., $\mathit{downacts}(\mathcal{R}) = \{\mathit{open}, \mathit{chooseL}, \mathit{chooseR}\}$
- $\mathit{upacts}(\mathcal{M})$: actions over which $\mathcal{M}$ synchronises with its parent, e.g., $\mathit{upacts}(\mathcal{M}_2) = \{\mathit{chooseL}, \mathit{chooseR}\}$
- $\mathit{locacts}(\mathcal{M})$: the remaining actions, e.g., $\mathit{locacts}(\mathcal{R}) = \{\mathit{beep}\}$

Tree Topologies and Live-reset Automata

Second restriction: all automata are live-reset



- $\mathcal{M}$ is live-reset if each synchronisation with its parent moves $\mathcal{M}$ to the initial state.

Tree Topologies and Live-reset Automata

Why these restrictions?

- **Tree topology**: allows recursive bottom-up reduction for subtrees.
- **Live-reset automata**: synchronisations are fire-and-forget, little additional bookkeeping needed.

Reduction for Trees of Height 1

- $\mathcal{G}$: a live-reset tree of height 1, with root $\mathcal{R}$ and $children(\mathcal{R}) = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$.
- We compute sum-of-squares product $\mathcal{SQ}(\mathcal{G})$ using only square products $\mathcal{R}||\mathcal{M}_i$ for $i \in \{1, \ldots, n\}$ (and some gadgets).
- $\mathcal{R}||\mathcal{M}_1|| \ldots ||\mathcal{M}_n \models EFp$ **iff** $\mathcal{SQ}(\mathcal{G}) \models EFp$

Reduction for Trees of Height 1, ct'd

$\mathcal{SQ}(\mathcal{G})$ (* We will build a new $\mathcal{LTS}$ $\mathcal{G}^r$. *)

1. Let $\mathit{states}(\mathcal{G}^r) = \emptyset$, $\mathit{transitions}(\mathcal{G}^r) = \emptyset$.
2. Make a fresh initial state s^0_{sq} and put it in $\mathit{states}(\mathcal{G}^r)$.
3. Put $\bigcup_{i=1}^{n} \mathit{states}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{states}(\mathcal{G}^r)$.
4. Put $\bigcup_{i=1}^{n} \mathit{transitions}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{transitions}(\mathcal{G}^r)$.
5. Via epsilon-transitions, connect s^0_{sq} with the initial state of each $\mathcal{R}||\mathcal{M}_i$, for $i \in \{1, \ldots, n\}$.
6. For each transition $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_i)$ in $\mathcal{R}||\mathcal{M}_i$ s.t. $act \in \mathit{upacts}(\mathcal{M}_i)$ add $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_j)$ to $\mathit{transitions}(\mathcal{G}^r)$, for all $j \in \{1, \ldots, n\}, i \neq j$.
7. Throw away from $\mathit{states}(\mathcal{G}^r)$ all the states from which no state with the root's action enabled can be reached.
8. Return $\mathcal{G}^r$.

Reduction for Trees of Height 1, ct'd

$\mathcal{SQ}(\mathcal{G})$ (* We will build a new $\mathcal{LTS}$ $\mathcal{G}^r$. *)

1. Let $\mathit{states}(\mathcal{G}^r) = \emptyset$, $\mathit{transitions}(\mathcal{G}^r) = \emptyset$.
2. Make a fresh initial state s^0_{sq} and put it in $\mathit{states}(\mathcal{G}^r)$.
3. Put $\bigcup_{i=1}^{n} \mathit{states}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{states}(\mathcal{G}^r)$.
4. Put $\bigcup_{i=1}^{n} \mathit{transitions}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{transitions}(\mathcal{G}^r)$.
5. Via epsilon-transitions, connect s^0_{sq} with the initial state of each $\mathcal{R}||\mathcal{M}_i$, for $i \in \{1, \ldots, n\}$.
6. For each transition $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_i)$ in $\mathcal{R}||\mathcal{M}_i$ s.t. $act \in \mathit{upacts}(\mathcal{M}_i)$ add $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_j)$ to $\mathit{transitions}(\mathcal{G}^r)$, for all $j \in \{1, \ldots, n\}, i \neq j$.
7. Throw away from $\mathit{states}(\mathcal{G}^r)$ all the states from which no state with the root's action enabled can be reached.
8. Return $\mathcal{G}^r$.

Reduction for Trees of Height 1, ct'd

$\mathcal{SQ}(\mathcal{G})$ (* We will build a new $\mathcal{LTS}$ $\mathcal{G}^r$. *)

1. Let $\mathit{states}(\mathcal{G}^r) = \emptyset$, $\mathit{transitions}(\mathcal{G}^r) = \emptyset$.
2. Make a fresh initial state s^0_{sq} and put it in $\mathit{states}(\mathcal{G}^r)$.
3. Put $\bigcup_{i=1}^{n} \mathit{states}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{states}(\mathcal{G}^r)$.
4. Put $\bigcup_{i=1}^{n} \mathit{transitions}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{transitions}(\mathcal{G}^r)$.
5. Via epsilon-transitions, connect s^0_{sq} with the initial state of each $\mathcal{R}||\mathcal{M}_i$, for $i \in \{1, \ldots, n\}$.
6. For each transition $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_i)$ in $\mathcal{R}||\mathcal{M}_i$ s.t. $act \in \mathit{upacts}(\mathcal{M}_i)$ add $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_j)$ to $\mathit{transitions}(\mathcal{G}^r)$, for all $j \in \{1, \ldots, n\}, i \neq j$.
7. Throw away from $\mathit{states}(\mathcal{G}^r)$ all the states from which no state with the root's action enabled can be reached.
8. Return $\mathcal{G}^r$.

Reduction for Trees of Height 1, ct'd

$\mathcal{SQ}(\mathcal{G})$ (* We will build a new $\mathcal{LTS}$ $\mathcal{G}^r$. *)

1. Let $\mathit{states}(\mathcal{G}^r) = \emptyset$, $\mathit{transitions}(\mathcal{G}^r) = \emptyset$.
2. Make a fresh initial state s^0_{sq} and put it in $\mathit{states}(\mathcal{G}^r)$.
3. Put $\bigcup_{i=1}^{n} \mathit{states}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{states}(\mathcal{G}^r)$.
4. Put $\bigcup_{i=1}^{n} \mathit{transitions}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{transitions}(\mathcal{G}^r)$.
5. Via epsilon-transitions, connect s^0_{sq} with the initial state of each $\mathcal{R}||\mathcal{M}_i$, for $i \in \{1, \ldots, n\}$.
6. For each transition $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_i)$ in $\mathcal{R}||\mathcal{M}_i$ s.t. $act \in \mathit{upacts}(\mathcal{M}_i)$ add $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_j)$ to $\mathit{transitions}(\mathcal{G}^r)$, for all $j \in \{1, \ldots, n\}, i \neq j$.
7. Throw away from $\mathit{states}(\mathcal{G}^r)$ all the states from which no state with the root's action enabled can be reached.
8. Return $\mathcal{G}^r$.

Reduction for Trees of Height 1, ct'd

$\mathcal{SQ}(\mathcal{G})$ (* We will build a new $\mathcal{LTS}$ $\mathcal{G}^r$. *)

1. Let $\mathit{states}(\mathcal{G}^r) = \emptyset$, $\mathit{transitions}(\mathcal{G}^r) = \emptyset$.
2. Make a fresh initial state s^0_{sq} and put it in $\mathit{states}(\mathcal{G}^r)$.
3. Put $\bigcup_{i=1}^{n} \mathit{states}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{states}(\mathcal{G}^r)$.
4. Put $\bigcup_{i=1}^{n} \mathit{transitions}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{transitions}(\mathcal{G}^r)$.
5. Via epsilon-transitions, connect s^0_{sq} with the initial state of each $\mathcal{R}||\mathcal{M}_i$, for $i \in \{1, \ldots, n\}$.
6. For each transition $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_i)$ in $\mathcal{R}||\mathcal{M}_i$ s.t. $act \in \mathit{upacts}(\mathcal{M}_i)$ add $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_j)$ to $\mathit{transitions}(\mathcal{G}^r)$, for all $j \in \{1, \ldots, n\}, i \neq j$.
7. Throw away from $\mathit{states}(\mathcal{G}^r)$ all the states from which no state with the root's action enabled can be reached.
8. Return $\mathcal{G}^r$.

Reduction for Trees of Height 1, ct'd

$\mathcal{SQ}(\mathcal{G})$ (* We will build a new $\mathcal{LTS}$ $\mathcal{G}^r$. *)

1. Let $\mathit{states}(\mathcal{G}^r) = \emptyset$, $\mathit{transitions}(\mathcal{G}^r) = \emptyset$.
2. Make a fresh initial state s^0_{sq} and put it in $\mathit{states}(\mathcal{G}^r)$.
3. Put $\bigcup_{i=1}^{n} \mathit{states}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{states}(\mathcal{G}^r)$.
4. Put $\bigcup_{i=1}^{n} \mathit{transitions}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{transitions}(\mathcal{G}^r)$.
5. Via epsilon-transitions, connect s^0_{sq} with the initial state of each $\mathcal{R}||\mathcal{M}_i$, for $i \in \{1, \ldots, n\}$.
6. For each transition $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_i)$ in $\mathcal{R}||\mathcal{M}_i$ s.t. $act \in \mathit{upacts}(\mathcal{M}_i)$ add $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_j)$ to $\mathit{transitions}(\mathcal{G}^r)$, for all $j \in \{1, \ldots, n\}, i \neq j$.
7. Throw away from $\mathit{states}(\mathcal{G}^r)$ all the states from which no state with the root's action enabled can be reached.
8. Return $\mathcal{G}^r$.

Reduction for Trees of Height 1, ct'd

$\mathcal{SQ}(\mathcal{G})$ (* We will build a new $\mathcal{LTS}$ $\mathcal{G}^r$. *)

1. Let $\mathit{states}(\mathcal{G}^r) = \emptyset$, $\mathit{transitions}(\mathcal{G}^r) = \emptyset$.
2. Make a fresh initial state s^0_{sq} and put it in $\mathit{states}(\mathcal{G}^r)$.
3. Put $\bigcup_{i=1}^{n} \mathit{states}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{states}(\mathcal{G}^r)$.
4. Put $\bigcup_{i=1}^{n} \mathit{transitions}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{transitions}(\mathcal{G}^r)$.
5. Via epsilon-transitions, connect s^0_{sq} with the initial state of each $\mathcal{R}||\mathcal{M}_i$, for $i \in \{1, \ldots, n\}$.
6. For each transition $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_i)$ in $\mathcal{R}||\mathcal{M}_i$ s.t. $act \in \mathit{upacts}(\mathcal{M}_i)$ add $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_j)$ to $\mathit{transitions}(\mathcal{G}^r)$, for all $j \in \{1, \ldots, n\}, i \neq j$.
7. Throw away from $\mathit{states}(\mathcal{G}^r)$ all the states from which no state with the root's action enabled can be reached.
8. Return $\mathcal{G}^r$.

Reduction for Trees of Height 1, ct'd

$\mathcal{SQ}(\mathcal{G})$ (* We will build a new $\mathcal{LTS}$ $\mathcal{G}^r$. *)

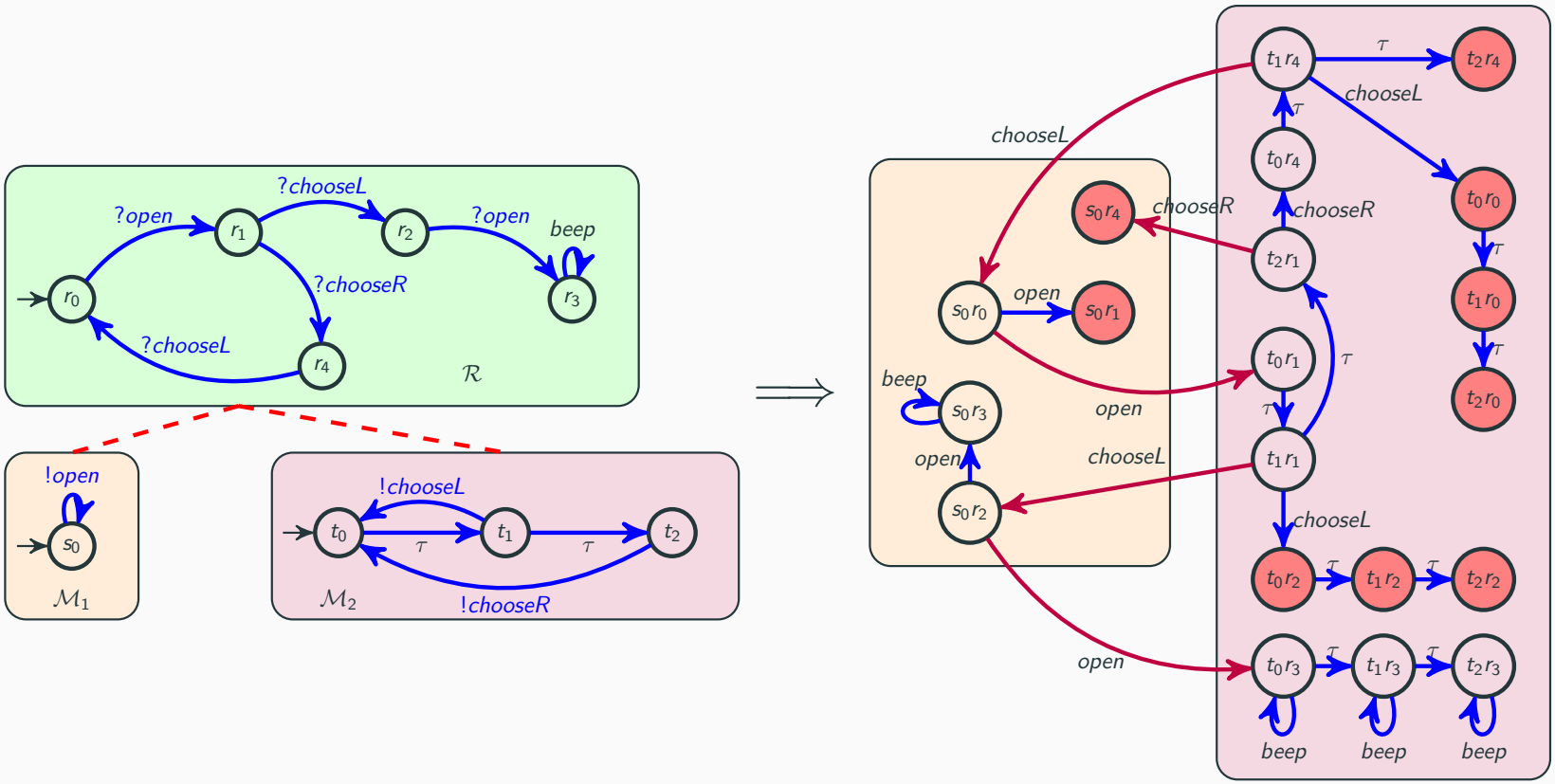
1. Let $\mathit{states}(\mathcal{G}^r) = \emptyset$, $\mathit{transitions}(\mathcal{G}^r) = \emptyset$.
2. Make a fresh initial state s^0_{sq} and put it in $\mathit{states}(\mathcal{G}^r)$.
3. Put $\bigcup_{i=1}^{n} \mathit{states}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{states}(\mathcal{G}^r)$.
4. Put $\bigcup_{i=1}^{n} \mathit{transitions}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{transitions}(\mathcal{G}^r)$.
5. Via epsilon-transitions, connect s^0_{sq} with the initial state of each $\mathcal{R}||\mathcal{M}_i$, for $i \in \{1, \ldots, n\}$.
6. For each transition $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_i)$ in $\mathcal{R}||\mathcal{M}_i$ s.t. $act \in \mathit{upacts}(\mathcal{M}_i)$ add $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_j)$ to $\mathit{transitions}(\mathcal{G}^r)$, for all $j \in \{1, \ldots, n\}, i \neq j$.
7. Throw away from $\mathit{states}(\mathcal{G}^r)$ all the states from which no state with the root's action enabled can be reached.
8. Return $\mathcal{G}^r$.

Reduction for Trees of Height 1, ct'd

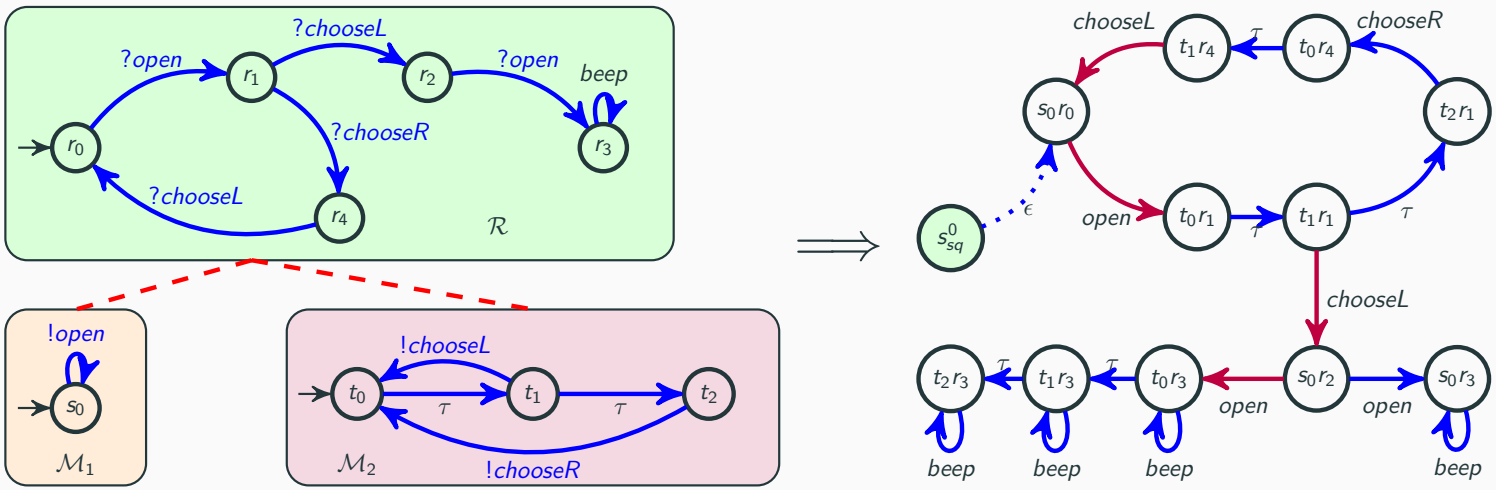
$\mathcal{SQ}(\mathcal{G})$ (* We will build a new $\mathcal{LTS}$ $\mathcal{G}^r$. *)

1. Let $\mathit{states}(\mathcal{G}^r) = \emptyset$, $\mathit{transitions}(\mathcal{G}^r) = \emptyset$.
2. Make a fresh initial state s^0_{sq} and put it in $\mathit{states}(\mathcal{G}^r)$.
3. Put $\bigcup_{i=1}^{n} \mathit{states}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{states}(\mathcal{G}^r)$.
4. Put $\bigcup_{i=1}^{n} \mathit{transitions}(\mathcal{R}||\mathcal{M}_i)$ in $\mathit{transitions}(\mathcal{G}^r)$.
5. Via epsilon-transitions, connect s^0_{sq} with the initial state of each $\mathcal{R}||\mathcal{M}_i$, for $i \in \{1, \ldots, n\}$.
6. For each transition $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_i)$ in $\mathcal{R}||\mathcal{M}_i$ s.t. $act \in \mathit{upacts}(\mathcal{M}_i)$ add $(s_{root}, s_i) \xrightarrow{act}_{sq} (s'_{root}, s^0_j)$ to $\mathit{transitions}(\mathcal{G}^r)$, for all $j \in \{1, \ldots, n\}, i \neq j$.
7. Throw away from $\mathit{states}(\mathcal{G}^r)$ all the states from which no state with the root's action enabled can be reached.
8. Return $\mathcal{G}^r$.

Reduction for Trees of Height 1: Example



Reduction for Trees of Height 1: Example



Notes and Limitations

- Generalisation for trees of any size: easy, omitted here.

Notes and Limitations

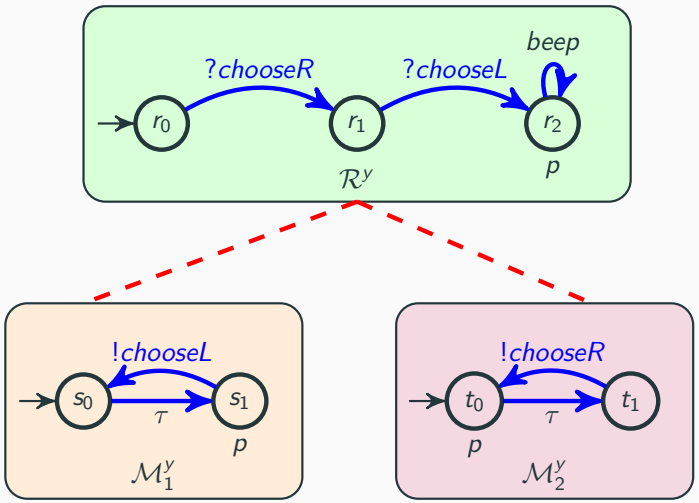
- Generalisation for trees of any size: easy, omitted here.
- If a tree sync. topology has n components, each of size size m, then the asynchronous product can reach size m^n. The size of the sum-of-squares product is at most $(n-1) \cdot m^2$.

Notes and Limitations

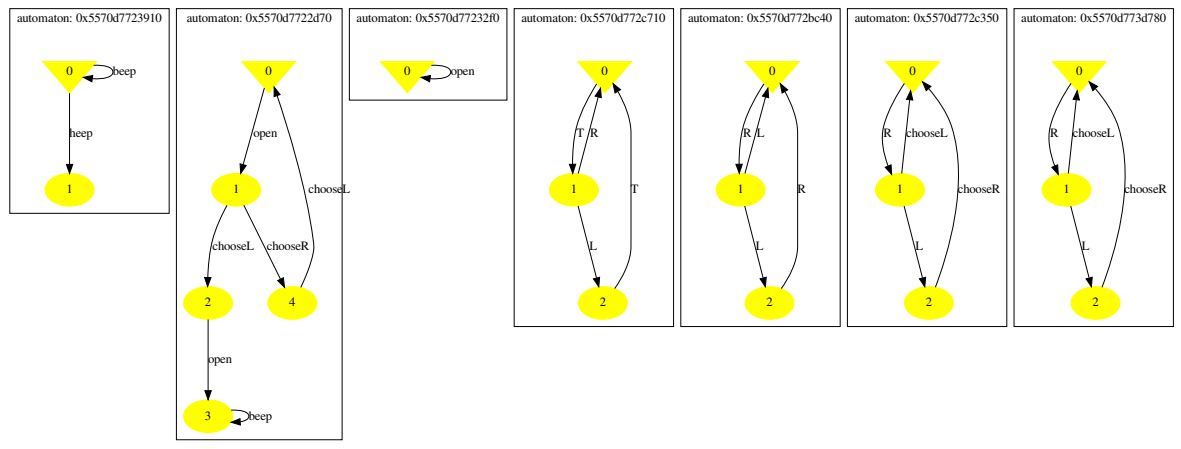
- Generalisation for trees of any size: easy, omitted here.
- If a tree sync. topology has n components, each of size size m, then the asynchronous product can reach size m^n. The size of the sum-of-squares product is at most $(n-1) \cdot m^2$.
- **Theorem:** $\mathcal{R}||\mathcal{M}_1|| \ldots ||\mathcal{M}_n \models EFp$ **iff** $\mathcal{SQ}(\mathcal{G}) \models EFp$**.**

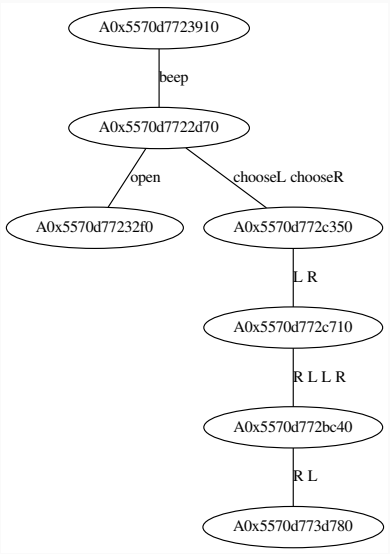
Notes and Limitations

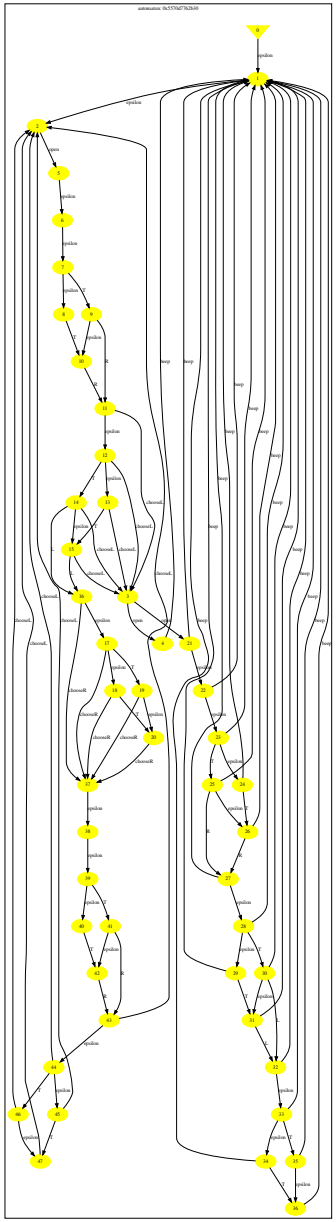
- Generalisation for trees of any size: easy, omitted here.
- If a tree sync. topology has n components, each of size size m, then the asynchronous product can reach size m^n. The size of the sum-of-squares product is at most $(n-1) \cdot m^2$.
- **Theorem:** $\mathcal{R}||\mathcal{M}_1|| \ldots ||\mathcal{M}_n \models EFp$ **iff** $\mathcal{SQ}(\mathcal{G}) \models EFp$**.**
- **But there is a sync. topology s.t.** $\mathcal{R}||\mathcal{M}_1|| \ldots ||\mathcal{M}_n \models EGp$ **and** $\mathcal{SQ}(\mathcal{G}) \not\models EGp$**.**



The Tool: Unreadable Screenshots







THANK YOU