Deciding Impossible Futures

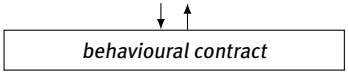
Maurice Laveaux and Tim Willemse
{m.laveaux,t.a.c.willemse}@tue.nl

TU/e

Motivation

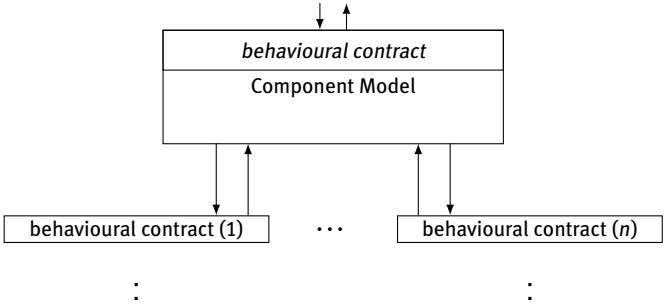
Motivation

Motivation



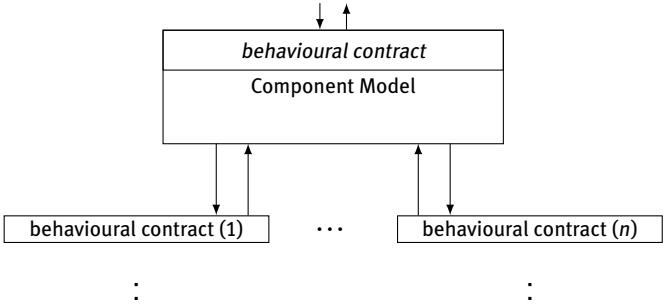
- System is designed by specifying its behavioural contract

Motivation



- System is designed by specifying its behavioural contract
- Contract is refined into component and other contracts and proven correct

Motivation



- System is designed by specifying its behavioural contract
- Contract is refined into component and other contracts and proven correct
- Generate executable code that is behaviourally equivalent to component model

Mauw's Three Stages of Maturity in Arts and Sciences

Mauw's Three Stages of Maturity in Arts and Sciences

Ideation

Mauw's Three Stages of Maturity in Arts and Sciences

Ideation

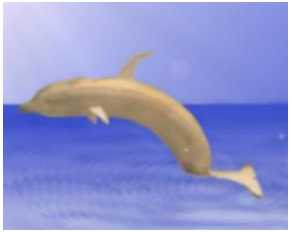
Prototyping

Mauw's Three Stages of Maturity in Arts and Sciences

Ideation

Prototyping

Consolidation



Impossible Futures

For states s and t, taken from two finite LTSs:

Impossible Futures

For states s and t, taken from two finite LTSs:

- Weak Trace inclusion: $s \sqsubseteq_{wt} t$ iff $\mathrm{WT}(s) \subseteq \mathrm{WT}(t)$

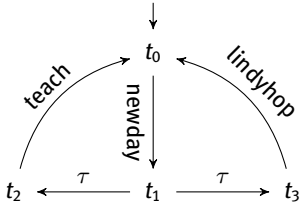
$$\mathrm{WT}(s) = \{w \in \mathrm{Act}^* \mid \exists s' : s \stackrel{w}{\Longrightarrow} s'\}$$

Impossible Futures

For states s and t, taken from two finite LTSs:

- Weak Trace inclusion: $s \sqsubseteq_{wt} t$ iff $\mathrm{WT}(s) \subseteq \mathrm{WT}(t)$

$$\mathrm{WT}(s) = \{w \in \mathrm{Act}^* \mid \exists s' : s \stackrel{w}{\Longrightarrow} s'\}$$

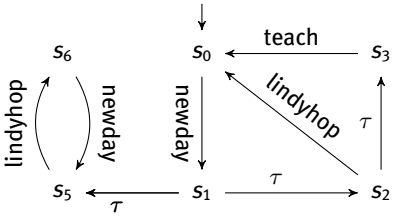


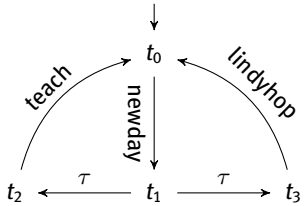
Impossible Futures

For states s and t, taken from two finite LTSs:

- Weak Trace inclusion: $s \sqsubseteq_{wt} t$ iff $\mathrm{WT}(s) \subseteq \mathrm{WT}(t)$

$$\mathrm{WT}(s) = \{w \in \mathrm{Act}^* \mid \exists s' : s \stackrel{w}{\Longrightarrow} s'\}$$



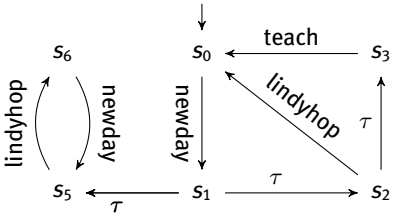


Impossible Futures

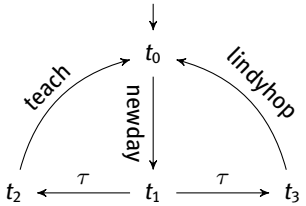
For states s and t, taken from two finite LTSs:

- Weak Trace inclusion: $s \sqsubseteq_{wt} t$ iff $\mathsf{WT}(s) \subseteq \mathsf{WT}(t)$

$$\mathsf{WT}(s) = \{w \in \mathsf{Act}^* \mid \exists s' : s \stackrel{w}{\Longrightarrow} s'\}$$



$\stackrel{?}{\sqsubseteq_{wt}}$



Impossible Futures

For states s and t, taken from two finite LTSs:

- Weak Trace inclusion: $s \sqsubseteq_{wt} t$ iff $\mathrm{WT}(s) \subseteq \mathrm{WT}(t)$

$$\mathrm{WT}(s) = \{w \in \mathrm{Act}^* \mid \exists s' : s \stackrel{w}{\Longrightarrow} s'\}$$

Impossible Futures

For states s and t, taken from two finite LTSs:

- Weak Trace inclusion: $s \sqsubseteq_{wt} t$ iff $\mathrm{WT}(s) \subseteq \mathrm{WT}(t)$

  $\mathrm{WT}(s) = \{w \in \mathrm{Act}^* \mid \exists s' : s \stackrel{w}{\Longrightarrow} s'\}$

- Impossible Future inclusion: $s \sqsubseteq_{if} t$ iff $\mathrm{IF}(s) \subseteq \mathrm{IF}(t)$ *(Voorhoeve&Mauw)*

  $\mathrm{IF}(s) = \{(w, X) \in \mathrm{Act}^* \times 2^{\mathrm{Act}^*} \mid \exists s' : s \stackrel{w}{\Longrightarrow} s' \text{ and } X \cap \mathrm{WT}(s') = \emptyset\}$

A Mystery

HOW TO EFFECTIVELY DECIDE $s \sqsubseteq_{if} t$?

A Detour: Weak Simulation

A Detour: Weak Simulation

A relation $R \subseteq S_1 \times S_2$ is a weak simulation iff whenever $s \; R \; t$ and $\alpha \in \text{Act} \cup \{\tau\}$:

A Detour: Weak Simulation

A relation $R \subseteq S_1 \times S_2$ is a weak simulation iff whenever $s \; R \; t$ and $\alpha \in \text{Act} \cup \{\tau\}$:

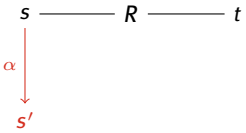
If $s \xrightarrow{\alpha} s'$ *for some* s'*, then there is some* t'
such that $t \stackrel{\overline{\alpha}}{\Longrightarrow} t'$ *and* $s' \; R \; t'$

s —— R —— t

A Detour: Weak Simulation

A relation $R \subseteq S_1 \times S_2$ is a weak simulation iff whenever $s \; R \; t$ and $\alpha \in \mathrm{Act} \cup \{\tau\}$:

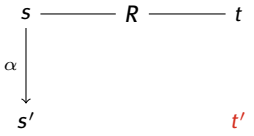
If $s \xrightarrow{\alpha} s'$ *for some* s'*, then there is some* t'
such that $t \stackrel{\overline{\alpha}}{\Longrightarrow} t'$ *and* $s' \; R \; t'$



A Detour: Weak Simulation

A relation $R \subseteq S_1 \times S_2$ is a weak simulation iff whenever $s \; R \; t$ and $\alpha \in \text{Act} \cup \{\tau\}$:

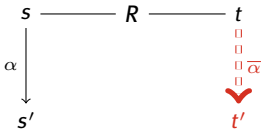
If $s \xrightarrow{\alpha} s'$ *for some* s'*, then there is some* t'
such that $t \stackrel{\overline{\alpha}}{\Longrightarrow} t'$ *and* $s' \; R \; t'$



A Detour: Weak Simulation

A relation $R \subseteq S_1 \times S_2$ is a weak simulation iff whenever $s \; R \; t$ and $\alpha \in \mathrm{Act} \cup \{\tau\}$:

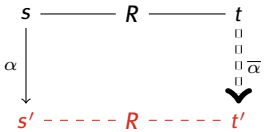
If $s \xrightarrow{\alpha} s'$ *for some* s'*, then there is some* t'
such that $t \stackrel{\overline{\alpha}}{\Longrightarrow} t'$ *and* $s' \; R \; t'$



A Detour: Weak Simulation

A relation $R \subseteq S_1 \times S_2$ is a weak simulation iff whenever $s \; R \; t$ and $\alpha \in \mathrm{Act} \cup \{\tau\}$:

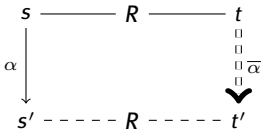
If $s \xrightarrow{\alpha} s'$ *for some* s'*, then there is some* t'
such that $t \stackrel{\overline{\alpha}}{\Longrightarrow} t'$ *and* $s' \; R \; t'$



A Detour: Weak Simulation

A relation $R \subseteq S_1 \times S_2$ is a weak simulation iff whenever $s\ R\ t$ and $\alpha \in \mathrm{Act} \cup \{\tau\}$:

If $s \xrightarrow{\alpha} s'$ *for some* s'*, then there is some* t'
such that $t \stackrel{\overline{\alpha}}{\Longrightarrow} t'$ *and* $s'\ R\ t'$



State s is weakly simulated by t, written $s \leq t$ iff there is a weak simulation R such that $s\ R\ t$

A Detour: Weak Simulation Games

A weak simulation game is a game played by Attacker and Defender on a graph $(V_A \cup V_D, E)$, where:

- Attacker owns vertices $V_A = S_1 \times S_2$, and Defender owns $V_D = S_1 \times S_2 \times (\mathrm{Act}_\tau \times S_1)$

A Detour: Weak Simulation Games

A weak simulation game is a game played by Attacker and Defender on a graph $(V_A \cup V_D, E)$, where:

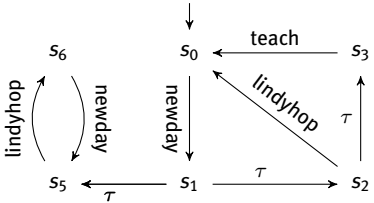
- Attacker owns vertices $V_A = S_1 \times S_2$, and Defender owns $V_D = S_1 \times S_2 \times (\mathrm{Act}_\tau \times S_1)$
- E is the least set satisfying:
  - if $s \xrightarrow{\alpha} s'$ then $(s,t)\ E\ (s,t,(\alpha,s'))$, and

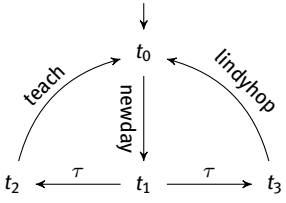
A Detour: Weak Simulation Games

A weak simulation game is a game played by Attacker and Defender on a graph $(V_A \cup V_D, E)$, where:

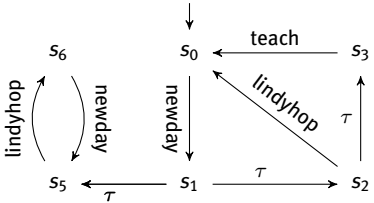
- Attacker owns vertices $V_A = S_1 \times S_2$, and Defender owns $V_D = S_1 \times S_2 \times (\mathrm{Act}_\tau \times S_1)$
- E is the least set satisfying:
  - if $s \xrightarrow{\alpha} s'$ then $(s,t)\ E\ (s,t,(\alpha,s'))$, and
  - if $t \xRightarrow{\bar{\alpha}} t'$ then $(s,t,(\alpha,s'))\ E\ (s',t')$

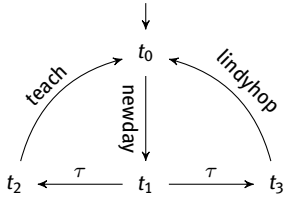
A Detour: Weak Simulation Games





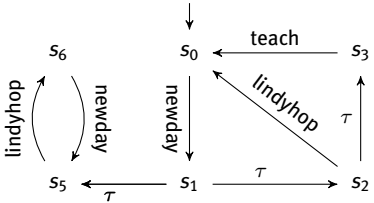
A Detour: Weak Simulation Games

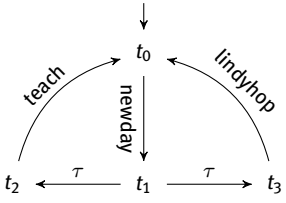


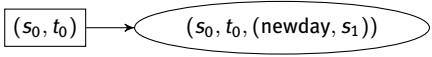


(s_0, t_0)

A Detour: Weak Simulation Games

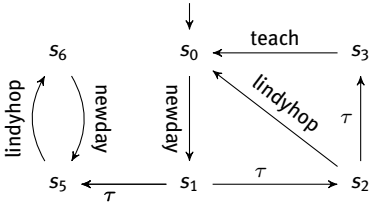


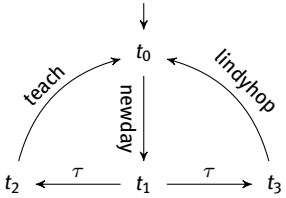


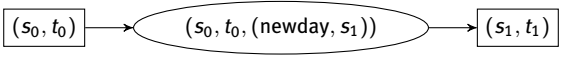


TU/e

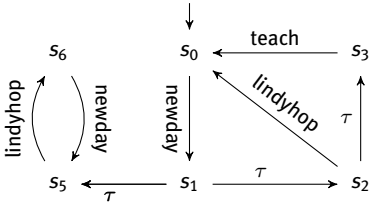
A Detour: Weak Simulation Games

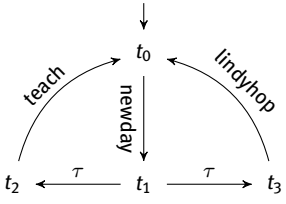


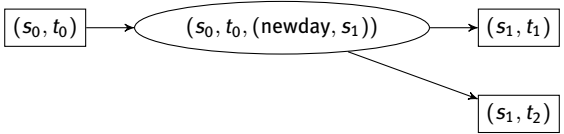




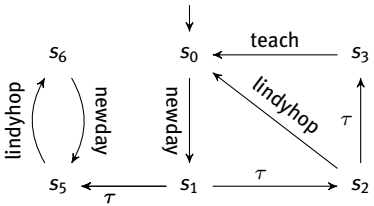
A Detour: Weak Simulation Games

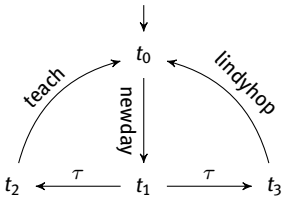


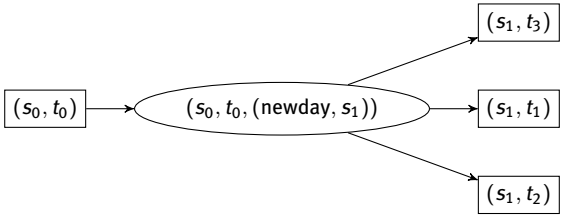




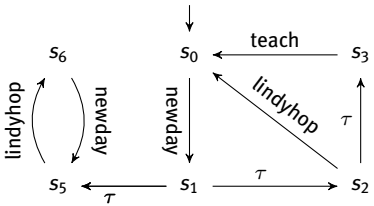
A Detour: Weak Simulation Games

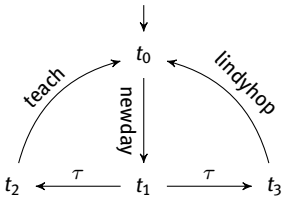


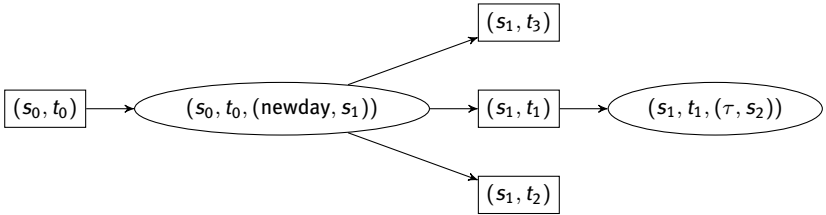




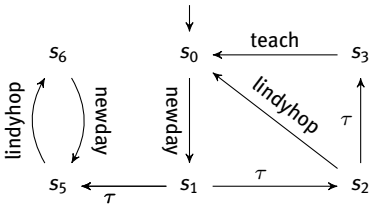
A Detour: Weak Simulation Games

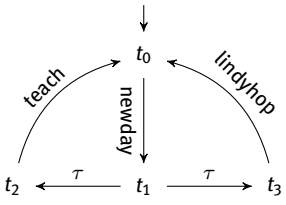


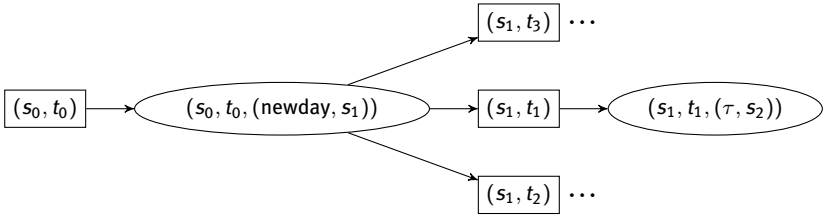




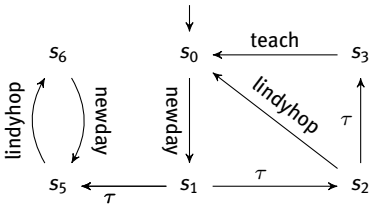
A Detour: Weak Simulation Games

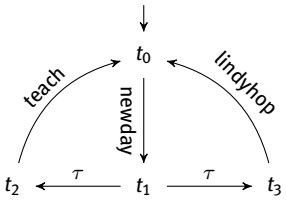


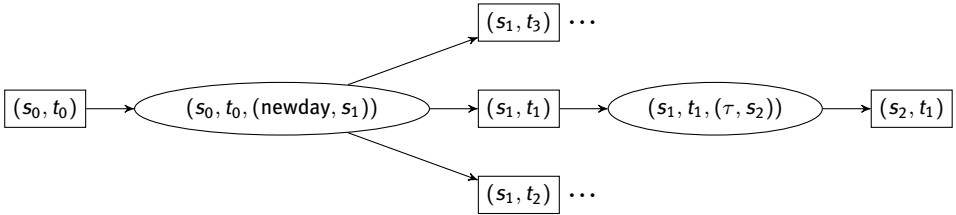




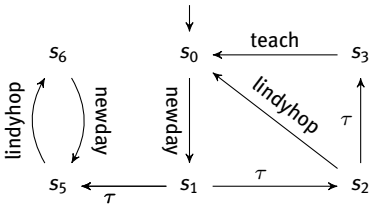
A Detour: Weak Simulation Games

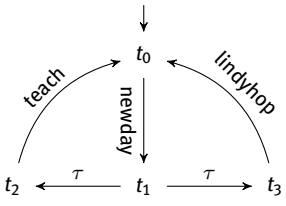


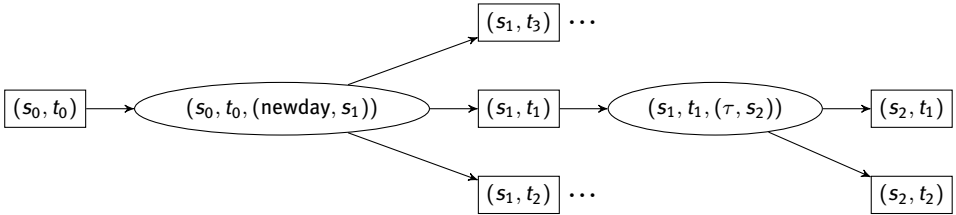




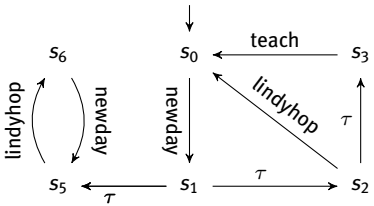
A Detour: Weak Simulation Games

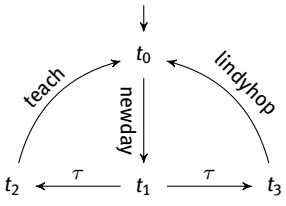


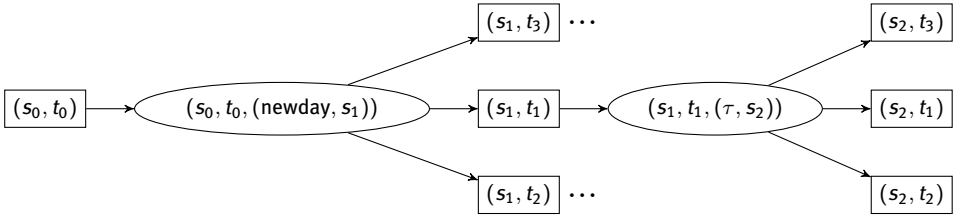




A Detour: Weak Simulation Games

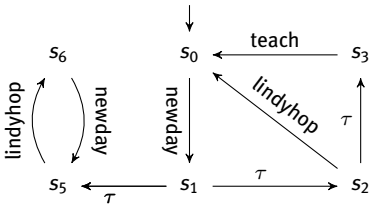


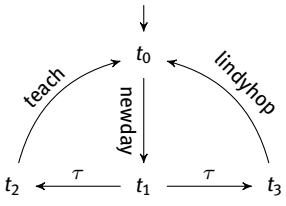


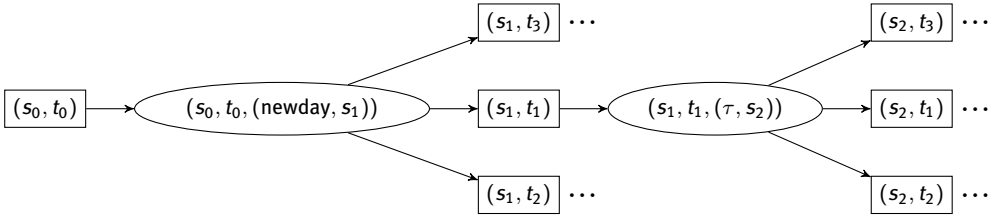


TU/e

A Detour: Weak Simulation Games







A Detour: Weak Simulation Games

- Attacker only wins plays in which Defender gets stuck; Defender wins all other plays

A Detour: Weak Simulation Games

- Attacker only wins plays in which Defender gets stuck; Defender wins all other plays
- $s \leq t$ if and only if Defender has a strategy to win every play starting in (s, t)

A Detour: Weak Simulation Games

- Attacker only wins plays in which Defender gets stuck; Defender wins all other plays
- $s \leq t$ if and only if Defender has a strategy to win every play starting in (s, t)

Crucial for us: for specifications t that are deterministic and concrete,

- $s \leq t$ if and only if $s \sqsubseteq_{wt} t$

A Detour: Weak Simulation Games

- Attacker only wins plays in which Defender gets stuck; Defender wins all other plays
- $s \leq t$ if and only if Defender has a strategy to win every play starting in (s, t)

Crucial for us: for specifications t that are deterministic and concrete,

- $s \leq t$ if and only if $s \sqsubseteq_{wt} t$
- Defender vertices have at most one successor

A Detour: Weak Simulation Games

- Attacker only wins plays in which Defender gets stuck; Defender wins all other plays
- $s \leq t$ if and only if Defender has a strategy to win every play starting in (s, t)

Crucial for us: for specifications t that are deterministic and concrete,

- $s \leq t$ if and only if $s \sqsubseteq_{wt} t$
- Defender vertices have at most one successor
- Hence, Attacker wins a vertex if and only if she can reach a Defender sink vertex

A Detour: Weak Simulation Games

- Attacker only wins plays in which Defender gets stuck; Defender wins all other plays
- $s \leq t$ if and only if Defender has a strategy to win every play starting in (s, t)

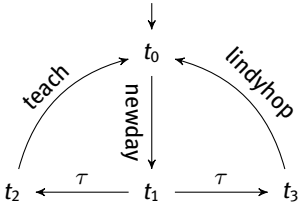
Crucial for us: for specifications t that are deterministic and concrete,

- $s \leq t$ if and only if $s \sqsubseteq_{wt} t$
- Defender vertices have at most one successor
- Hence, Attacker wins a vertex if and only if she can reach a Defender sink vertex
- So in this case $s \sqsubseteq_{wt} t$ is a reachability problem solvable using a breadth-first or depth-first search.

An equivalent deterministic & concrete LTS

The normal form norm(s) of a state s yields a deterministic, concrete LTS satisfying WT(s) = WT(norm(s))

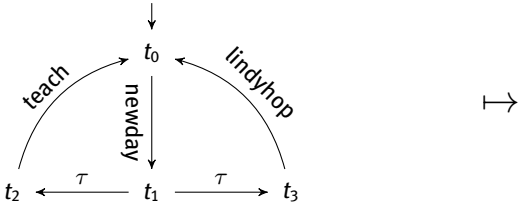
Example



An equivalent deterministic & concrete LTS

The normal form norm(s) of a state s yields a deterministic, concrete LTS satisfying $\mathrm{WT}(s) = \mathrm{WT}(\mathrm{norm}(s))$

Example

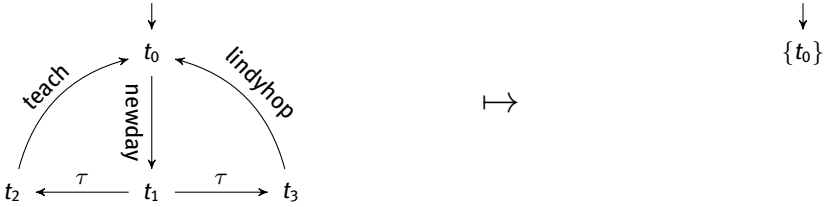


$\mapsto$

An equivalent deterministic & concrete LTS

The normal form norm(s) of a state s yields a deterministic, concrete LTS satisfying $\mathrm{WT}(s) = \mathrm{WT}(\mathrm{norm}(s))$

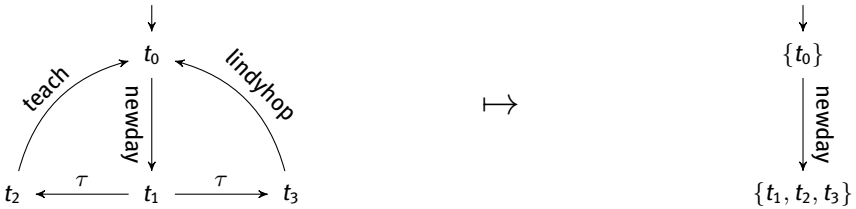
Example



An equivalent deterministic & concrete LTS

The normal form norm(s) of a state s yields a deterministic, concrete LTS satisfying WT(s) = WT(norm(s))

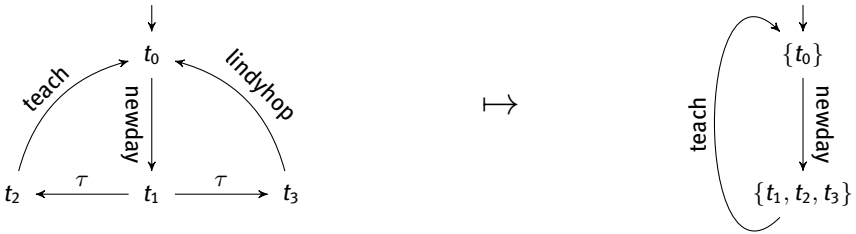
Example



An equivalent deterministic & concrete LTS

The normal form norm(s) of a state s yields a deterministic, concrete LTS satisfying WT(s) = WT(norm(s))

Example

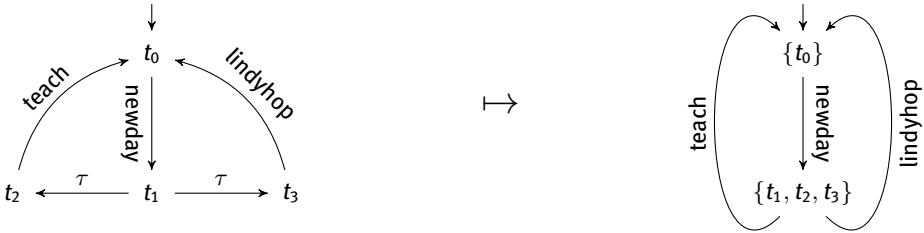


TU/e

An equivalent deterministic & concrete LTS

The normal form norm(s) of a state s yields a deterministic, concrete LTS satisfying WT(s) = WT(norm(s))

Example



Putting some of these thoughts together

So:

$$s \sqsubseteq_{wt} t$$

if and only if

$$s \sqsubseteq_{wt} \mathrm{norm}(t)$$

if and only if

$$s \leq \mathrm{norm}(t)$$

if and only if

No Defender-sink is reachable from $(s, \mathrm{norm}(t))$

How to Decide Impossible Futures?

Modifying the simulation game to decide Impossible Futures:

- Note that in the simulation game for $s \leq \text{norm}(t)$, Attacker vertices are of the shape (s, U)

How to Decide Impossible Futures?

Modifying the simulation game to decide Impossible Futures:

- Note that in the simulation game for $s \leq \text{norm}(t)$, Attacker vertices are of the shape (s, U)
- In vertex (s, U), it 'suffices' for Attacker to check $v \sqsubseteq_{wt} s$ for every $v \in U$ ..... possibly $|U|$ new games

How to Decide Impossible Futures?

Modifying the simulation game to decide Impossible Futures:

- Note that in the simulation game for $s \leq \text{norm}(t)$, Attacker vertices are of the shape (s, U)
- In vertex (s, U), it 'suffices' for Attacker to check $v \sqsubseteq_{wt} s$ for every $v \in U$ ..... possibly $|U|$ new games
- Stronger winning condition for Attacker: she also wins plays passing through vertices (s, U) for which:

  $v \sqsubseteq_{wt} s$ fails for every $v \in U$

How to Decide Impossible Futures?

Modifying the simulation game to decide Impossible Futures:

- Note that in the simulation game for $s \leq \text{norm}(t)$, Attacker vertices are of the shape (s, U)
- In vertex (s, U), it 'suffices' for Attacker to check $v \sqsubseteq_{wt} s$ for every $v \in U$ ..... possibly $|U|$ new games
- Stronger winning condition for Attacker: she also wins plays passing through vertices (s, U) for which:

  $v \sqsubseteq_{wt} s$ fails for every $v \in U$

- Defender wins $(s, \text{norm}(t))$ in this modified simulation game iff $s \sqsubseteq_{if} t$

How to Decide Impossible Futures?

Modifying the simulation game to decide Impossible Futures:

- Note that in the simulation game for $s \leq \text{norm}(t)$, Attacker vertices are of the shape (s, U)
- In vertex (s, U), it 'suffices' for Attacker to check $v \sqsubseteq_{wt} s$ for every $v \in U$ ..... possibly $|U|$ new games
- Stronger winning condition for Attacker: she also wins plays passing through vertices (s, U) for which:

  $v \sqsubseteq_{wt} s$ fails for every $v \in U$

- Defender wins $(s, \text{norm}(t))$ in this modified simulation game iff $s \sqsubseteq_{if} t$

But: implemented naively this approach is sloooooooooooooow and does not even terminate in many cases

Antichains

The practical runtime for checking $\sqsubseteq_{wt}$ and $\sqsubseteq_{if}$ can be improved:

Antichains

The practical runtime for checking $\sqsubseteq_{wt}$ and $\sqsubseteq_{if}$ can be improved:

- Define $\preccurlyeq$ on Attacker vertices as follows: $(s, U) \preccurlyeq (s', U')$ iff $s = s'$ and $U \subseteq U'$

Antichains

The practical runtime for checking $\sqsubseteq_{wt}$ and $\sqsubseteq_{if}$ can be improved:

- Define $\preccurlyeq$ on Attacker vertices as follows: $(s, U) \preccurlyeq (s', U')$ iff $s = s'$ and $U \subseteq U'$
- Nice property: if Attacker wins (s, U'), then she also wins all (s, U) smaller than (s, U')

Antichains

The practical runtime for checking $\sqsubseteq_{wt}$ and $\sqsubseteq_{if}$ can be improved:

- Define $\preccurlyeq$ on Attacker vertices as follows: $(s, U) \preccurlyeq (s', U')$ iff $s = s'$ and $U \subseteq U'$
- Nice property: if Attacker wins (s, U'), then she also wins all (s, U) smaller than (s, U')
- Consequence: once we visit (s, U), we need not further explore any (s, U') larger than (s, U)

Antichains

Optimising the search for Defender-sinks in the simulation game:

- Store visited Attacker vertices in an antichain $A \subseteq S_1 \times 2^{S_2} \setminus \{\emptyset\}$
  A is an antichain when neither $x \preccurlyeq y$ nor $y \preccurlyeq x$ for all $x \neq y$ in A

Antichains

Optimising the search for Defender-sinks in the simulation game:

- Store visited Attacker vertices in an antichain $A \subseteq S_1 \times 2^{S_2} \setminus \{\emptyset\}$
  A is an antichain when neither $x \preccurlyeq y$ nor $y \preccurlyeq x$ for all $x \neq y$ in A
- Antichain A over-approximates all discovered vertices: $\mathrm{UP}(A) = \{(s, U') \mid \exists (s, U) \in A : (s, U) \preccurlyeq (s, U')\}$

Antichains

Optimising the search for Defender-sinks in the simulation game:

- Store visited Attacker vertices in an antichain $A \subseteq S_1 \times 2^{S_2} \setminus \{\emptyset\}$
  A is an antichain when neither $x \preccurlyeq y$ nor $y \preccurlyeq x$ for all $x \neq y$ in A
- Antichain A over-approximates all discovered vertices: $\mathrm{UP}(A) = \{(s, U') \mid \exists (s, U) \in A : (s, U) \preccurlyeq (s, U')\}$

When encountering a new vertex (s, U) while exploring the simulation game:

Antichains

Optimising the search for Defender-sinks in the simulation game:

- Store visited Attacker vertices in an antichain $A \subseteq S_1 \times 2^{S_2} \setminus \{\emptyset\}$
  A is an antichain when neither $x \preccurlyeq y$ nor $y \preccurlyeq x$ for all $x \neq y$ in A
- Antichain A over-approximates all discovered vertices: $\mathrm{UP}(A) = \{(s, U') \mid \exists (s, U) \in A : (s, U) \preccurlyeq (s, U')\}$

When encountering a new vertex (s, U) while exploring the simulation game:

- If $(s, U) \in \mathrm{UP}(A)$, ignore (s, U);

Antichains

Optimising the search for Defender-sinks in the simulation game:

- Store visited Attacker vertices in an antichain $A \subseteq S_1 \times 2^{S_2} \setminus \{\emptyset\}$
  A is an antichain when neither $x \preccurlyeq y$ nor $y \preccurlyeq x$ for all $x \neq y$ in A
- Antichain A over-approximates all discovered vertices: $\mathrm{UP}(A) = \{(s, U') \mid \exists (s, U) \in A : (s, U) \preccurlyeq (s, U')\}$

When encountering a new vertex (s, U) while exploring the simulation game:

- If $(s, U) \in \mathrm{UP}(A)$, ignore (s, U);
- Otherwise, add (s, U) to the *todo* set, and update antichain A:

$$A, \mathit{todo} := (A \setminus \mathrm{UP}(\{\,(s, U)\,\})) \cup \{(s, U)\},\ \mathit{todo} \cup \{(s, U)\}$$

Antichains

Optimising the search for Defender-sinks in the simulation game:

- Store visited Attacker vertices in an antichain $A \subseteq S_1 \times 2^{S_2} \setminus \{\emptyset\}$
  A *is an antichain when neither* $x \preccurlyeq y$ *nor* $y \preccurlyeq x$ *for all* $x \neq y$ *in* A
- Antichain A over-approximates all discovered vertices: $\mathrm{UP}(A) = \{(s, U') \mid \exists (s, U) \in A : (s, U) \preccurlyeq (s, U')\}$

When encountering a new vertex (s, U) while exploring the simulation game:

- If $(s, U) \in \mathrm{UP}(A)$, ignore (s, U);
- Otherwise, add (s, U) to the *todo* set, and update antichain A:

$$A, \mathit{todo} := (A \setminus \mathrm{UP}(\{\,(s, U)\,\})) \cup \{(s, U)\},\ \mathit{todo} \cup \{(s, U)\}$$

But: implemented naively this approach is also slow though it typically terminates

Further Optimisations for Impossible Futures and Conclusions

- Speed-up 1: memorise and reuse the antichains of successful $v \sqsubseteq_{wt} s$ checks carried out for (s, U)

Further Optimisations for Impossible Futures and Conclusions

- Speed-up 1: memorise and reuse the antichains of successful $v \sqsubseteq_{wt} s$ checks carried out for (s, U)

  yields up-to an additional 15-fold performance improvement in some cases

Further Optimisations for Impossible Futures and Conclusions

- Speed-up 1: memorise and reuse the antichains of successful $v \sqsubseteq_{wt} s$ checks carried out for (s, U)

  yields up-to an additional 15-fold performance improvement in some cases

- Speed-up 2: on convergent paths in s, only check additional $v \sqsubseteq_{wt} s$ checks when s is 'stable'

Further Optimisations for Impossible Futures and Conclusions

- Speed-up 1: memorise and reuse the antichains of successful $v \sqsubseteq_{wt} s$ checks carried out for (s, U)

  yields up-to an additional 15-fold performance improvement in some cases

- Speed-up 2: on convergent paths in s, only check additional $v \sqsubseteq_{wt} s$ checks when s is 'stable'

  yields up-to an additional 50-fold performance improvement in some cases

Further Optimisations for Impossible Futures and Conclusions

- Speed-up 1: memorise and reuse the antichains of successful $v \sqsubseteq_{wt} s$ checks carried out for (s, U)

  yields up-to an additional 15-fold performance improvement in some cases

- Speed-up 2: on convergent paths in s, only check additional $v \sqsubseteq_{wt} s$ checks when s is 'stable'

  yields up-to an additional 50-fold performance improvement in some cases

- Speed-up 3: maintain a (reversed) antichain preserving information about failed $v \sqsubseteq_{wt} s$ checks

Further Optimisations for Impossible Futures and Conclusions

- Speed-up 1: memorise and reuse the antichains of successful $v \sqsubseteq_{wt} s$ checks carried out for (s, U)

  yields up-to an additional 15-fold performance improvement in some cases

- Speed-up 2: on convergent paths in s, only check additional $v \sqsubseteq_{wt} s$ checks when s is 'stable'

  yields up-to an additional 50-fold performance improvement in some cases

- Speed-up 3: maintain a (reversed) antichain preserving information about failed $v \sqsubseteq_{wt} s$ checks

  yields neglible performance improvements

Further Optimisations for Impossible Futures and Conclusions

- Speed-up 1: memorise and reuse the antichains of successful $v \sqsubseteq_{wt} s$ checks carried out for (s, U)
  yields up-to an additional 15-fold performance improvement in some cases
- Speed-up 2: on convergent paths in s, only check additional $v \sqsubseteq_{wt} s$ checks when s is 'stable'
  yields up-to an additional 50-fold performance improvement in some cases
- Speed-up 3: maintain a (reversed) antichain preserving information about failed $v \sqsubseteq_{wt} s$ checks
  yields neglible performance improvements

Combining all three can yield up-to a 100-fold performance improvement

Further Optimisations for Impossible Futures and Conclusions

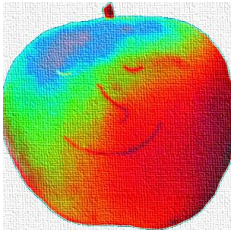
- Speed-up 1: memorise and reuse the antichains of successful $v \sqsubseteq_{wt} s$ checks carried out for (s, U)
  yields up-to an additional 15-fold performance improvement in some cases
- Speed-up 2: on convergent paths in s, only check additional $v \sqsubseteq_{wt} s$ checks when s is 'stable'
  yields up-to an additional 50-fold performance improvement in some cases
- Speed-up 3: maintain a (reversed) antichain preserving information about failed $v \sqsubseteq_{wt} s$ checks
  yields neglible performance improvements

Combining all three can yield up-to a 100-fold performance improvement

Towards consolidation: Verum is currently investigating how to integrate Impossible Futures in their Dezyne tool

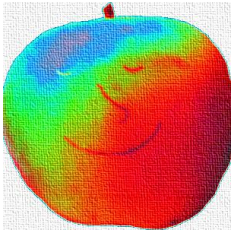
Sjouke's Newton Moments and ...

Portrait of an Artist 1

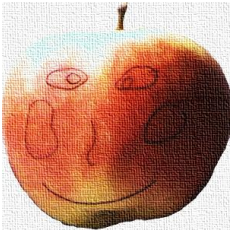


Sjouke's Newton Moments and ...

Portrait of an Artist 1

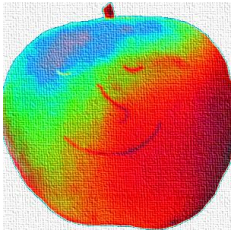


Portrait of an Artist 2

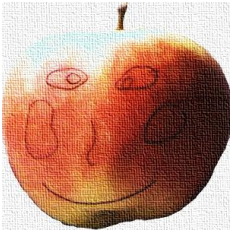


Sjouke's Newton Moments and ...

Portrait of an Artist 1



Portrait of an Artist 2



Artist informally known as Sjouke who created both self-portraits

Sjouke's Newton Moments and Footage of Attack Trees