CAAL
Concurrency Workbench Aalborg Edition

Jesper R. Andersen Nicklas Andersen
Søren Enevoldsen Mathias M. Hansen Kim G. Larsen
Simon R. Olesen Jiri Srba Jacob K. Wortmann

Department of Computer Science
Aalborg University

CASSTING Meeting, Cachan, October, 2015

MOTIVATION

- At Aalborg University we teach a courses on reactive systems and concurrency theory, based on Milner's CCS.

Existing tools for analysing processes described in CCS.

- Concurrency Workbench (CWB)
- Concurrency Workbench New Century (CWB-NC)
- Tool for the Analysis of Process Algebras (TAPAs)
- pseuCo (Java to CCS)
- CCS in Maude and in Haskell (experiential)

RELATED WORK

There exist various other tools, typically using a superclass of CCS, of industrial strength. To mention some:

- FDR3
- mCRL2
- CADP
- LTSmin
- PRISM

CAAL lacks the power of those above, but focuses on

- usability,
- use of graphical aids, and
- aims at use in education.

RELATED WORK

There exist various other tools, typically using a superclass of CCS, of industrial strength. To mention some:

- FDR3
- mCRL2
- CADP
- LTSmin
- PRISM

CAAL lacks the power of those above, but focuses on

- usability,
- use of graphical aids, and
- aims at use in education.

CCS SYNTAX

The collection of CCS expressions $\mathcal{P}$ is given by the grammar:

$$P, Q ::= K \;\Big|\; \alpha.P \;\Big|\; P + Q \;\Big|\; P \mid Q \;\Big|\; P[f] \;\Big|\; P \setminus L \;\Big|\; 0$$

where K is a process constant (name), α is an action, L a set of labels, and f is a relabelling function.

- A process is guarded if all occurrences of process constants are under the scope of action prefixing.

Example of semantics (SOS rules):

$$a.b.0 + c.0 \xrightarrow{a} b.0$$

$$a.b.0 \mid c.0 \xrightarrow{a} b.0 \mid c.0$$

$$a.0 \mid \overline{a}.0 \xrightarrow{\tau} 0 \mid 0$$

CCS SYNTAX

The collection of CCS expressions $\mathcal{P}$ is given by the grammar:

$$P, Q ::= K \;\Big|\; \alpha.P \;\Big|\; P + Q \;\Big|\; P \mid Q \;\Big|\; P[f] \;\Big|\; P \setminus L \;\Big|\; 0$$

where K is a process constant (name), α is an action, L a set of labels, and f is a relabelling function.

- A process is guarded if all occurrences of process constants are under the scope of action prefixing.

Example of semantics (SOS rules):

$$a.b.0 + c.0 \xrightarrow{a} b.0$$

$$a.b.0 \mid c.0 \xrightarrow{a} b.0 \mid c.0$$

$$a.0 \mid \bar{a}.0 \xrightarrow{\tau} 0 \mid 0$$

STRONG BISIMULATION

Definition (Bisimulation)

A binary relation $\mathcal{R}$ over the set of states of an LTS is a bisimulation if and only if whenever $s \mathcal{R} t$ and α is an action:

$$\text{if } s \xrightarrow{\alpha} s' \text{ then } t \xrightarrow{\alpha} t' \text{ s.t. } s' \mathcal{R} t', \text{ and}$$
$$\text{if } t \xrightarrow{\alpha} t' \text{ then } s \xrightarrow{\alpha} s' \text{ s.t. } s' \mathcal{R} t'.$$

Two states s and t are *strongly bisimilar*, written $s \sim t$, iff there is a bisimulation that relates them.

WEAK BISIMULATION

We write $s \stackrel{\alpha}{\Rightarrow} s'$ if

- $\alpha \neq \tau$ and $s(\stackrel{\tau}{\rightarrow})^* \circ \stackrel{\alpha}{\rightarrow} \circ (\stackrel{\tau}{\rightarrow})^* s'$ or
- $\alpha = \tau$ and $s(\stackrel{\tau}{\rightarrow}^*)s'$.

Definition (Weak Bisimulation)

A binary relation $\mathcal{R}$ over the set of states of an LTS is a weak bisimulation if and only if whenever $s \mathcal{R} t$ and α is an action:

$$\text{if } s \stackrel{\alpha}{\Rightarrow} s' \text{ then } t \stackrel{\alpha}{\Rightarrow} t' \text{ s.t. } s' \mathcal{R} t' \text{ and}$$
$$\text{If } t \stackrel{\alpha}{\Rightarrow} t' \text{ then } s \stackrel{\alpha}{\Rightarrow} s' \text{ s.t. } s' \mathcal{R} t'.$$

Two states s and t are *weakly bisimilar*, written $s \approx t$, iff there is a weak bisimulation that relates them.

BISIMULATION GAME

Given two states s and t:

- Attacker aims to show $s \not\sim t$; defender aims to show $s \sim t$.
- Configurations of the game are pairs (s, t).

1. Attacker selects a transition, $s \xrightarrow{\alpha} s'$ from s (or t).
2. Defender matches with transition from t (or s), $t \xrightarrow{\alpha} t'$.
3. The pair (s', t') forms next configuration and the round repeats.

- Attacker wins if defender is unable to match action.
- Defender wins if attacker is unable to select any action.
- Defender also wins if the play is infinite (or the same configuration has been seen twice).

Theorem

Attacker has a winning strategy from (s, t) iff $s \not\sim t$.

HML SYNTAX WITH RECURSIVE VARIABLES

$$\phi ::= tt \mid ff \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \langle\alpha\rangle\phi \mid [\alpha]\phi \mid \langle\langle\alpha\rangle\rangle\phi \mid [[\alpha]]\phi \mid X$$

where X is a variable declared as either a minimum fixed point

$$X \overset{\min}{=} \phi$$

or a maximum fixed point

$$X \overset{\max}{=} \phi\,.$$

- Note: CAAL allows for more than one variable but they may not be mutually recursive.

Example

$$X \overset{\max}{=} [a]X \wedge \langle b\rangle tt$$

HML-GAME

Given state s and formula ϕ:

- Attacker aims to show $s \not\models \phi$; defender aims to show $s \models \phi$.
- Configurations consist of process and formula pairs (s, ϕ).

- Attacker picks a successor from $\phi_1 \wedge \phi_2$, $[a]\phi_1$, or $[[a]]\phi_1$
- Defender picks a successor from $\phi_1 \vee \phi_2$, $\langle a \rangle \phi_1$, or $\langle\langle a \rangle\rangle \phi_1$
- In (s, X) the variable X unfolds to its definition.

- Attacker wins in (s, ff) or any infinite play in *min* context.
- Defender wins in (s, tt) or any infinite play in *max* context.
- Either player loses if stuck.

Theorem
Attacker has a universal winning strategy from (s, ϕ) iff $s \not\models \phi$.

DISTINGUISHING FORMULA

Theorem (Hennessy and Milner)

Let P and Q be guarded CCS processes. Then:

$$P \sim Q \text{ if and only if } P \models \phi \Leftrightarrow Q \models \phi$$

for all ϕ without the modalities $\langle\langle\alpha\rangle\rangle$ and $[[\alpha]]$. And

$$P \approx Q \text{ if and only if } P \models \phi \Leftrightarrow Q \models \phi$$

for all ϕ without the modalities $\langle\alpha\rangle$ and $[\alpha]$.

- Hence, if $s \not\sim t$ (or $s \not\approx t$) there is a distinguishing formula ϕ such that $s \models \phi$ and $t \not\models \phi$.

PROBLEMS

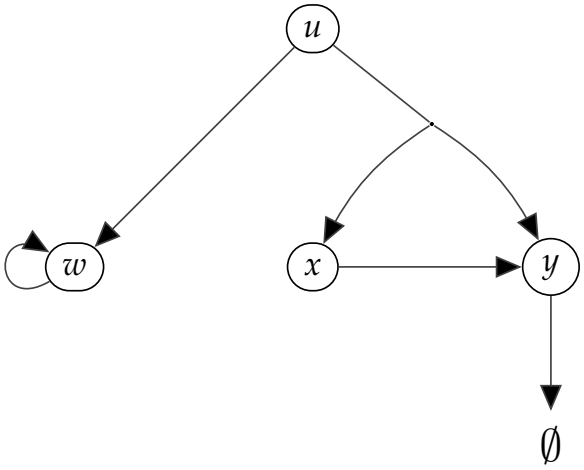
- Given two CCS processes, decide if they are strongly/weakly bisimilar.
- Given a CCS process P and a formula ϕ, decide if $P \models \phi$.
- If two processes are not bisimilar, find a distinguishing formula.
- If two processes are not bisimilar or a formula does not hold, play bisimulation/model checking games and win against the user.

DEPENDENCY GRAPHS

Definition (Dependency Graph)

A dependency graph is a pair (V, E) where V is a finite set of nodes and $E \subseteq V \times \mathcal{P}(V)$ is a finite set of hyperedges. A hyperedge is a pair (v, T) where v is the source and $T \subseteq V$ is the target set.

$$V = \{u, w, x, y\}, E = \{(y, \emptyset), (x, \{y\}), (u, \{x, y\}), (u, \{w\}), (w, \{w\})\}$$



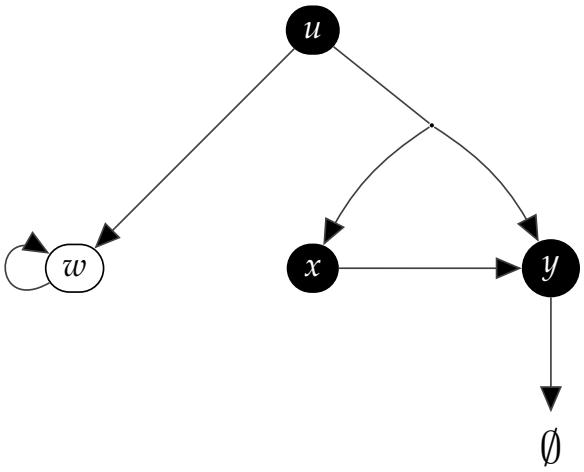
ASSIGNMENTS

Assignment is a function $A : V \to \{0, 1\}$.

Definition (Pre-fixed-point Assignment)

A *pre-fixed-point assignment* A satisfies that whenever $(v, T) \in E$ such that for all $v' \in T$ we have $A(v') = 1$ then also $A(v) = 1$.

There is a minimum pre-fixed-point assignment A_{min}.



ALGORITHMS FOR A_{min} [LIU AND SMOLKA]

- Global Iteration over all nodes.
  1. Initialize all nodes to 0.
  2. Relax the assignment by iterating over all edges until no change is needed.

- Local 'on-the-fly' algorithm.
  1. Explore the hyperedges in a forward and backward manner.
  2. Terminate as soon as the value for the source node is known.

ENCODING TO DEPENDENCY GRAPH

To determine whether $s \sim t$, $s \approx t$ or $s \models \phi$.

1. Construct a dependency graph rooted at v.
2. Compute $A_{min}(v)$.
3. The value $A_{min}(v)$ determines the result.

Remark: this approach can give conclusive results even for infinite-state systems (if we use the local algorithm).

ENCODING TO DEPENDENCY GRAPH

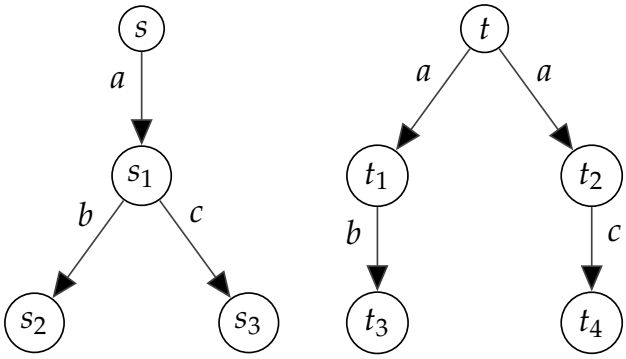
To determine whether $s \sim t$, $s \approx t$ or $s \models \phi$.

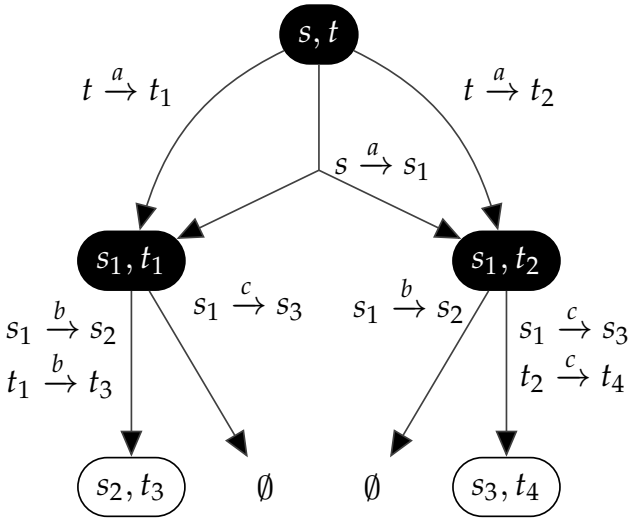
1. Construct a dependency graph rooted at v.
2. Compute $A_{min}(v)$.
3. The value $A_{min}(v)$ determines the result.

Remark: this approach can give conclusive results even for infinite-state systems (if we use the local algorithm).

BISIMULATION ENCODING

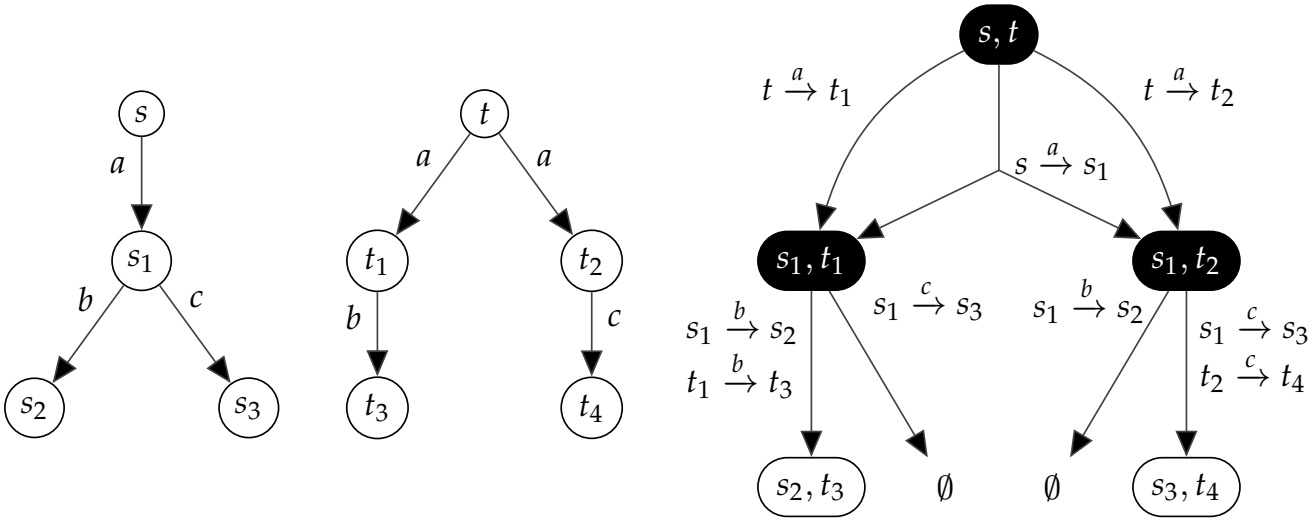
Determining whether $s \sim t$.





BISIMULATION ENCODING

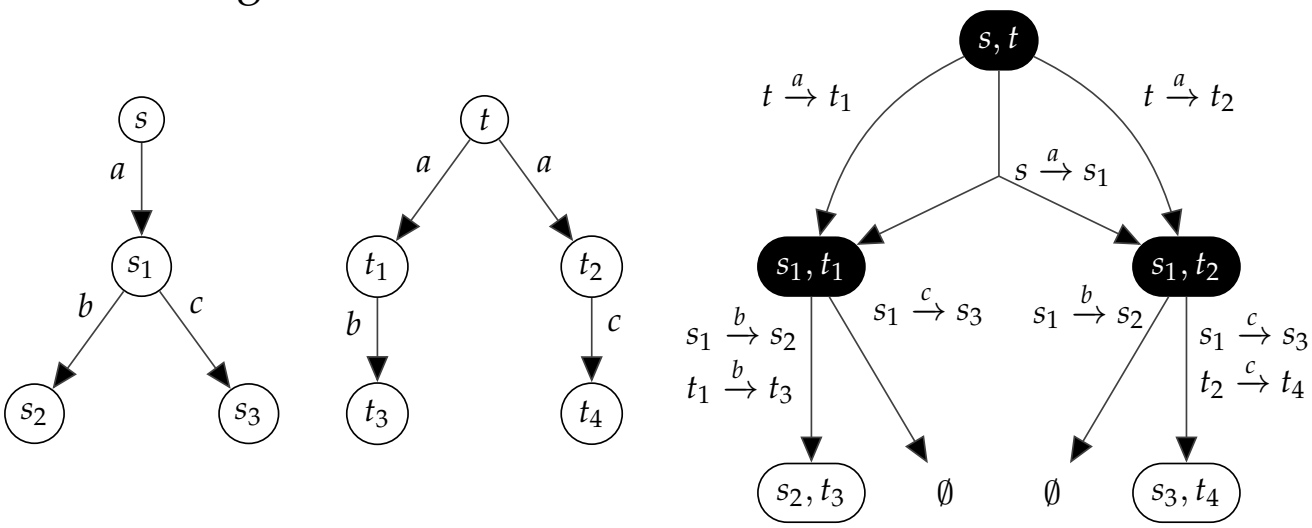
Determining whether $s \sim t$.



Same graph used by bisimulation game.

BISIMULATION ENCODING

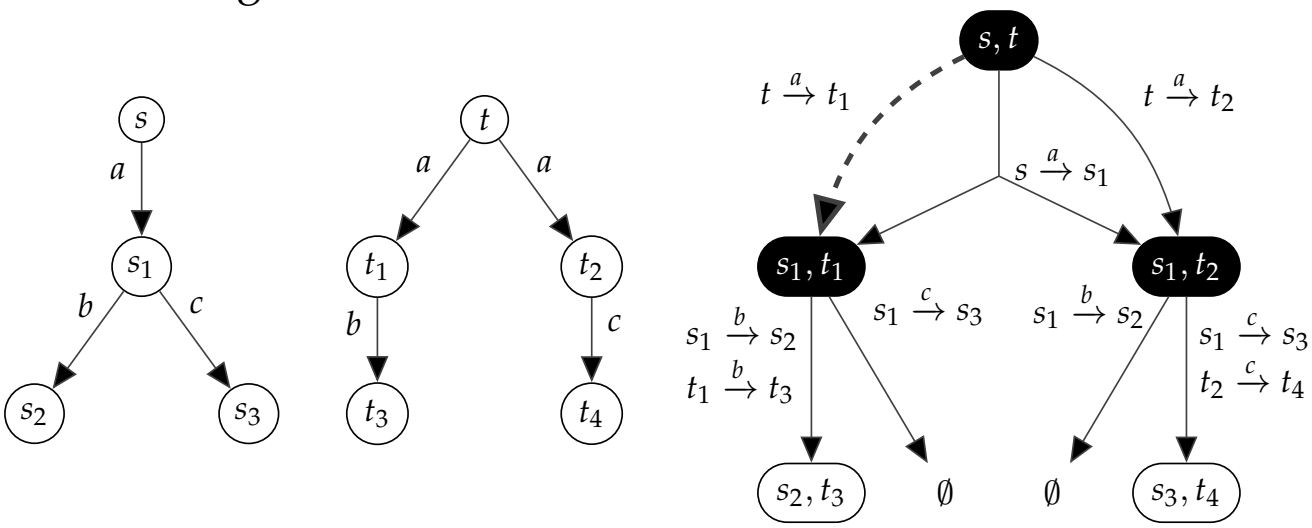
Determining whether $s \sim t$.



Distinguishing Formula: $\phi = DF((s,t))$
$s \models \phi$ and $t \not\models \phi$

BISIMULATION ENCODING

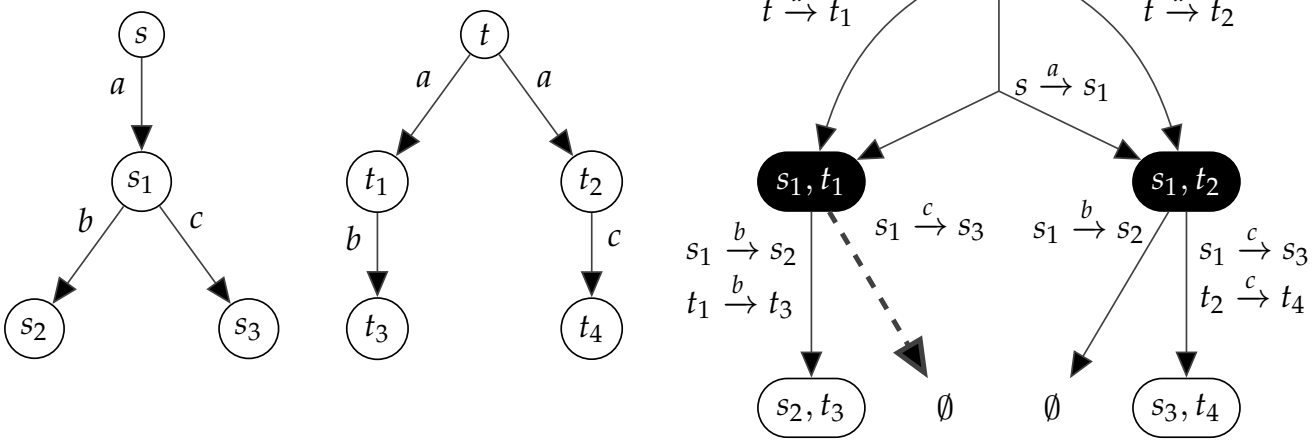
Determining whether $s \sim t$.



Distinguishing Formula: $\phi = [a]DF((s_1, t_1))$
$s \models \phi$ and $t \not\models \phi$

BISIMULATION ENCODING

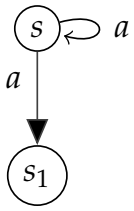
Determining whether $s \sim t$.



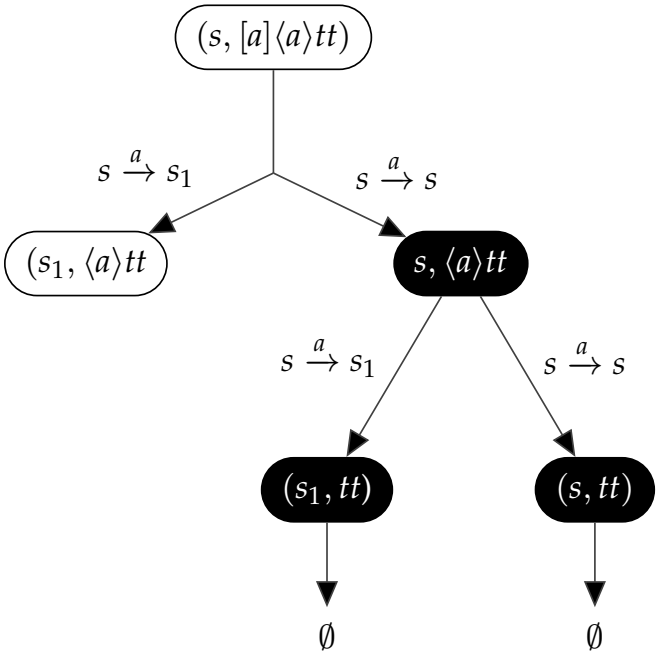
Distinguishing Formula: $\phi = [a]\langle c \rangle tt$
$s \models \phi$ and $t \not\models \phi$

MODEL CHECKING

Determining whether $s \models \phi$.

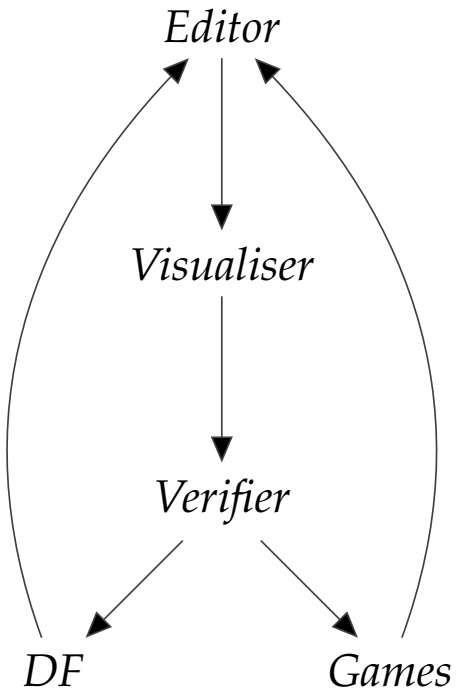


$$\phi = [a]\langle a\rangle tt$$



Same graph used by model checking game.

TOOL IMPLEMENTATION

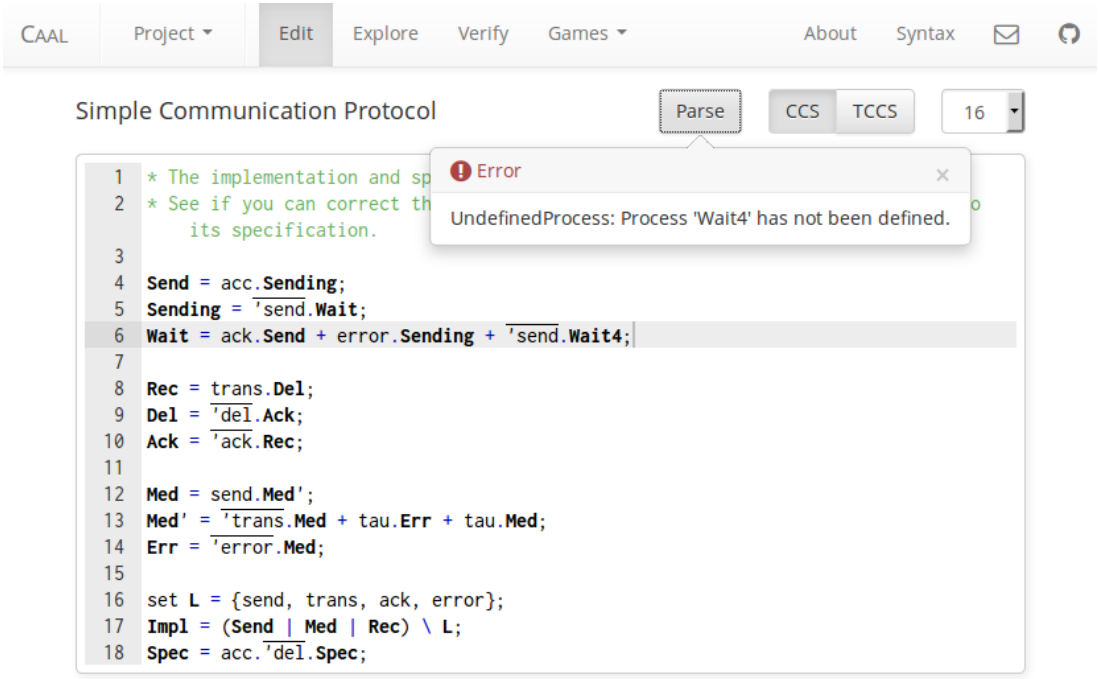


- Cross platform web based tool running in browser.
- Written in JavaScript.

`http://caal.cs.aau.dk`

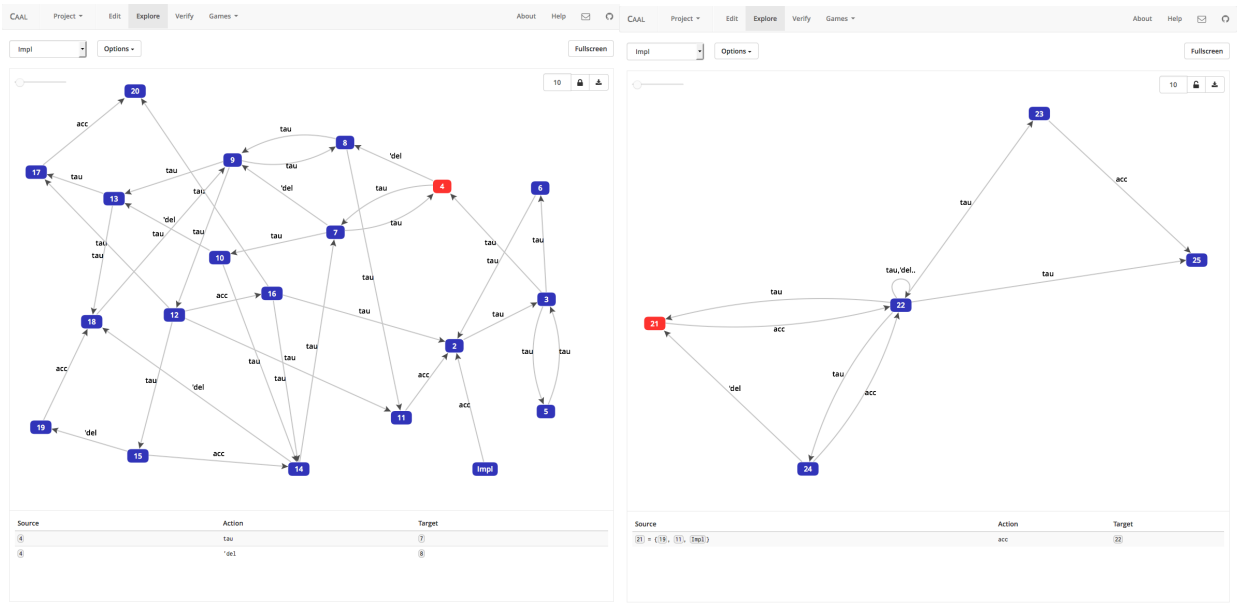
EDITOR

- Syntax highlighting for CCS and discrete timed CCS.



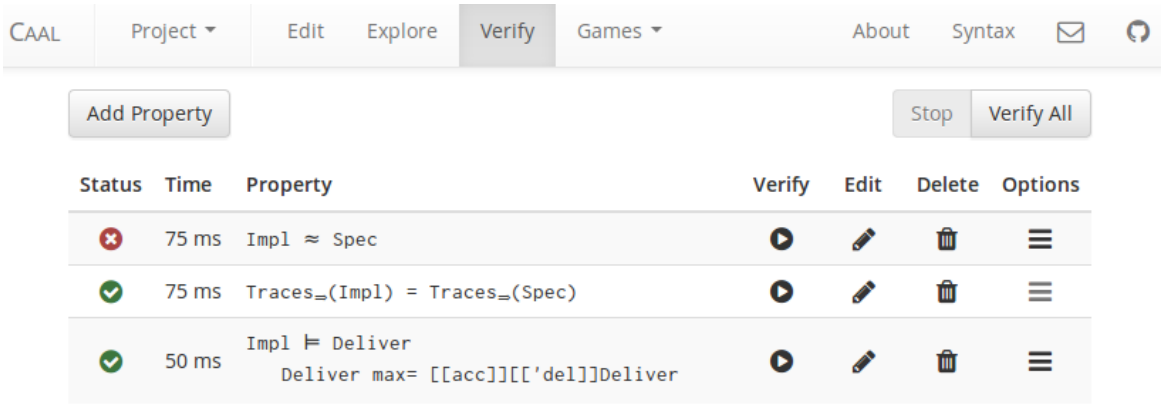
VISUALIZER

- Visualize labelled transition system. Custom state horizon.
- Bisimulation collapse.



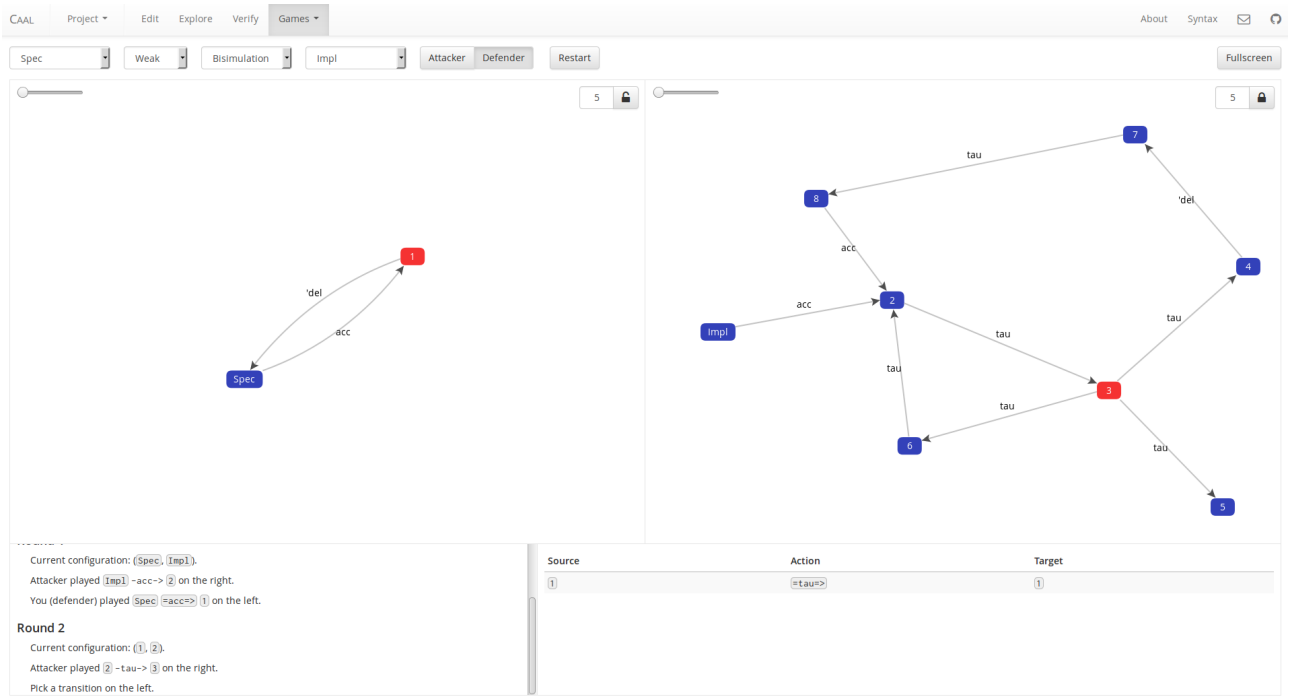
VERIFIER

- Preorders: trace inclusion and simulation.
- Equivalences: trace, simulation, and bisimulation.
- Both strong/weak and timed/untimed variants.
- Model checking of recursive HML formulae.
- Debugging options: distinguishing formulae, games.



GAMES

- Bisimulation and HML game.
- Play against a computer (and loose).



DEMO

CONCLUSION

CAAL is a graphical tool for the use in teaching about reactive systems.

- CAAL is trivial to run in any modern browser.
- Supports CCS and HML syntax including discrete time.
- Offers equivalence and model checking of CSS/TCCS.
- Focuses on usability rather than speed.
- Includes intuitive debugging using distinguishing formulae and games.

Current work.

- Verification-as-a-service.
- Performance optimisations.
- Parallelisation of fixed-point computation.

BIBLIOGRAPHY

[1] J.R. Andersen, M.M. Hansen, and N. Andersen. CAAL 2.0: Equivalences, preorders and games for CCS and TCCS. Master's thesis, Aalborg University, 2015.

[2] J.K. Wortmann, S.R. Olesen, and S. Enevoldsen. CAAL 2.0: Recursive HML, distinguishing formulae, equivalence collapses and parallel fixed-point computations. Master's thesis, Aalborg University, 2015.