Self assembling trees with RCCS

Fabien Tarissan

Joint work with Vincent Danos & Jean Krivine

Équipe PPS, Université Paris 7 & CNRS

INRIA-Rocquencourt, Université Paris 6

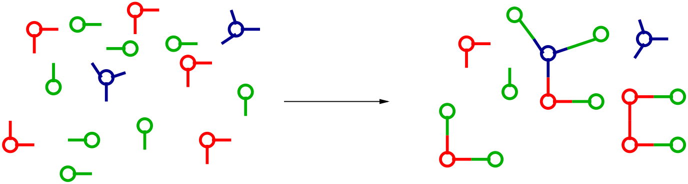
SOS – August 26, 2006

The problem

- Question : How a collective behaviour may emerge from elementary interactions (forward and backward)
- Applications
  - Molecular biology (backward)
  - Genetic engineering (forward)
  - Distributed robotics (forward)

The problem

- Question : How a collective behaviour may emerge from elementary interactions (forward and backward)
- Applications
  - Molecular biology (backward)
  - Genetic engineering (forward)
  - Distributed robotics (forward)

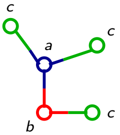


A syntax for the trees

- V : a set of nodes.
- $\delta : V \rightarrow \mathbb{N}$: degree map.

$$(v, \{t_1, \ldots, t_n\})$$

- $\mathrm{edg}(t, v)$: number of edges connected to v in t
- t is coherent $\iff \forall v \in \mathrm{n}(t), \delta(v) = \mathrm{edg}(t, v)$



$$\delta(a) = 3 \quad \delta(b) = 2 \quad \delta(c) = 1$$
$$(a, \{(b, \{c\}), c, c\})$$

An LTS for the specification

The specification : $\mathrm{SPEC} = (S, L, \rightarrow)$

- Element of the state space : $(N, \sum_i t_i)$ with $N \subseteq V$ and t_i coherent.
- Labels : set of coherent trees
- Transition relation : for all coherent tree t

$$(N + \mathrm{n}(t)\ ,\ \sum_i t_i) \rightarrow_t (N\ ,\ t + \sum_i t_i)$$

An LTS for the specification

The specification : $\mathrm{SPEC} = (S, L, \rightarrow)$

- Element of the state space : $(N, \sum_i t_i)$ with $N \subseteq V$ and t_i coherent.
- Labels : set of coherent trees
- Transition relation : for all coherent tree t

$$(N + \mathrm{n}(t)\ ,\ \sum_i t_i) \rightarrow_t (N\ ,\ t + \sum_i t_i)$$

We are looking for an implementation in a process algebra:

- Concurrency
- Binary interactions
- Mathematical tool for proving correctness.

Possible program

$$\begin{array}{lcl}
\mathrm{NODE}_i & \stackrel{def}{=} & \tau.\mathrm{BUILD}_{i\star}^{\delta(i),\emptyset} + \sum_{j\in I} r_{ij}.\mathrm{BUILD}_{ij}^{\delta(i),\emptyset} \\
\mathrm{BUILD}_{ij}^{n+1,S} & \stackrel{def}{=} & \sum_{k\in I} \overline{r}_{ik}.\mathrm{BUILD}_{ij}^{n,S\cup\{k\}} + kill_i.\mathrm{ABORT}_i^S \\
\mathrm{BUILD}_{i\star}^{n+1,S} & \stackrel{def}{=} & \sum_{k\in I} \overline{r}_{ik}.\mathrm{BUILD}_{ij}^{n,S\cup\{k\}} + \tau.\mathrm{ABORT}_i^S \\
\mathrm{BUILD}_{i\alpha}^{0,S} & \stackrel{def}{=} & \mathrm{WAIT}_{i\alpha}^{|S|,S} \\
\mathrm{WAIT}_{ij}^{n+1,S} & \stackrel{def}{=} & w_i.\mathrm{WAIT}_{ij}^{n,S} + kill_i.\mathrm{ABORT}_i^S \\
\mathrm{WAIT}_{i\star}^{n+1,S} & \stackrel{def}{=} & w_i.\mathrm{WAIT}_{i\star}^{n,S} + \tau.\mathrm{ABORT}_i^S \\
\mathrm{WAIT}_{ij}^{0,S} & \stackrel{def}{=} & \overline{w}_j.\uparrow_{ij}^S + kill_i.\mathrm{ABORT}_i^S \\
\mathrm{WAIT}_{i\star}^{0,S} & \stackrel{def}{=} & ok_i.\uparrow_{i\star}^S \\
\mathrm{FREE}^{S\cup\{i\}}(end) & \stackrel{def}{=} & \overline{kill}_i.\mathrm{FREE}^S(end) \qquad \uparrow_{ij}^S \stackrel{def}{=} \tau.\uparrow_{ij}^S + kill_i.\mathrm{ABORT}_{ij}^S \\
\mathrm{FREE}^{\emptyset}(end) & \stackrel{def}{=} & \overline{end}.0 \qquad \uparrow_{i\star}^S \stackrel{def}{=} \tau.\uparrow_{i\star}^S \\
\mathrm{ABORT}_i^S & \stackrel{def}{=} & (end)(\mathrm{FREE}^S(end) \mid end.\mathrm{NODE}_i)
\end{array}$$

Possible program

$$\mathrm{NODE}_i \overset{def}{=} \tau.\mathrm{BUILD}_{i\star}^{\delta(i),\emptyset} + \sum_{j\in I} r_{ij}.\mathrm{BUILD}_{ij}^{\delta(i),\emptyset}$$

$$\mathrm{BUILD}_{ij}^{n+1,S} \overset{def}{=} \sum_{k\in I} \overline{r}_{ik}.\mathrm{BUILD}_{ij}^{n,S\cup\{k\}} + kill_i.\mathrm{ABORT}_i^S$$

$$\mathrm{BUILD}_{i\star}^{n+1,S} \overset{def}{=} \sum_{k\in I} \overline{r}_{ik}.\mathrm{BUILD}_{ij}^{n,S\cup\{k\}} + \tau.\mathrm{ABORT}_i^S$$

$$\mathrm{BUILD}_{i\alpha}^{0,S} \overset{def}{=} \mathrm{WAIT}_{i\alpha}^{|S|,S}$$

$$\mathrm{WAIT}_{ij}^{n+1,S} \overset{def}{=} w_i.\mathrm{WAIT}_{ij}^{n,S} + kill_i.\mathrm{ABORT}_i^S$$

$$\mathrm{WAIT}_{i\star}^{n+1,S} \overset{def}{=} w_i.\mathrm{WAIT}_{i\star}^{n,S} + \tau.\mathrm{ABORT}_i^S$$

$$\mathrm{WAIT}_{ij}^{0,S} \overset{def}{=} \overline{w}_j.\uparrow_{ij}^S + kill_i.\mathrm{ABORT}_i^S$$

$$\mathrm{WAIT}_{i\star}^{0,S} \overset{def}{=} ok_i.\uparrow_{i\star}^S$$

$$\mathrm{FREE}^{S\cup\{i\}}(end) \overset{def}{=} \overline{kill}_i.\mathrm{FREE}^S(end) \qquad \uparrow_{ij}^S \overset{def}{=} \tau.\uparrow_{ij}^S + kill_i.\mathrm{ABORT}_{ij}^S$$

$$\mathrm{FREE}^{\emptyset}(end) \overset{def}{=} \overline{end}.0 \qquad \uparrow_{i\star}^S \overset{def}{=} \tau.\uparrow_{i\star}^S$$

$$\mathrm{ABORT}_i^S \overset{def}{=} (end)(\mathrm{FREE}^S(end) \mid end.\mathrm{NODE}_i)$$

Possible program

$$\mathrm{NODE}_i \stackrel{def}{=} \tau.\mathrm{BUILD}_{i\star}^{\delta(i),\emptyset} + \sum_{j\in I} r_{ij}.\mathrm{BUILD}_{ij}^{\delta(i),\emptyset}$$

$$\mathrm{BUILD}_{ij}^{n+1,S} \stackrel{def}{=} \sum_{k\in I} \overline{r}_{ik}.\mathrm{BUILD}_{ij}^{n,S\cup\{k\}} + kill_i.\mathrm{ABORT}_i^S$$

$$\mathrm{BUILD}_{i\star}^{n+1,S} \stackrel{def}{=} \sum_{k\in I} \overline{r}_{ik}.\mathrm{BUILD}_{ij}^{n,S\cup\{k\}} + \tau.\mathrm{ABORT}_i^S$$

$$\mathrm{BUILD}_{i\alpha}^{0,S} \stackrel{def}{=} \mathrm{WAIT}_{i\alpha}^{|S|,S}$$

$$\mathrm{WAIT}_{ij}^{n+1,S} \stackrel{def}{=} w_i.\mathrm{WAIT}_{ij}^{n,S} + kill_i.\mathrm{ABORT}_i^S$$

$$\mathrm{WAIT}_{i\star}^{n+1,S} \stackrel{def}{=} w_i.\mathrm{WAIT}_{i\star}^{n,S} + \tau.\mathrm{ABORT}_i^S$$

$$\mathrm{WAIT}_{ij}^{0,S} \stackrel{def}{=} \overline{w}_j.\uparrow_{ij}^S + kill_i.\mathrm{ABORT}_i^S$$

$$\mathrm{WAIT}_{i\star}^{0,S} \stackrel{def}{=} ok_i.\uparrow_{i\star}^S$$

$$\mathrm{FREE}^{S\cup\{i\}}(end) \stackrel{def}{=} \overline{kill}_i.\mathrm{FREE}^S(end) \qquad \uparrow_{ij}^S \stackrel{def}{=} \tau.\uparrow_{ij}^S + kill_i.\mathrm{ABORT}_{ij}^S$$

$$\mathrm{FREE}^{\emptyset}(end) \stackrel{def}{=} \overline{end}.0 \qquad \uparrow_{i\star}^S \stackrel{def}{=} \tau.\uparrow_{i\star}^S$$

$$\mathrm{ABORT}_i^S \stackrel{def}{=} (end)(\mathrm{FREE}^S(end) \mid end.\mathrm{N}_i)$$

What is required ?

The code is very complicated:

- equivalence with the specification
- more difficult to understand, reuse, patch, . . .

What is required ?

The code is very complicated:

- equivalence with the specification
- more difficult to understand, reuse, patch, ...

Our approach:

1. forward code in CCS
2. good properties w.r.t specification
3. lift into RCCS (nothing to do)
4. application of a theorem: RCCS term is equivalent to the specification

CCS : syntax

Actions:	α	$::=$	$a \mid \bar{a}$	Action on a channel
			$\mid \tau$	Silent action
Processes:	p	$::=$	0	End of process
			$\mid \sum \alpha_i . p_i$	Guarded Choice
			$\mid (p \parallel p)$	Fork
			$\mid (a)p$	Restriction
			$\mid \mathrm{D}(\tilde{x})$	Definition

CCS : semantics

$$\frac{}{\sum_i \alpha_i.p_i \rightarrow_{\alpha_i} p_i}\ (\mathrm{act}) \qquad \frac{p \rightarrow_\alpha p'}{p \parallel q \rightarrow_\alpha p' \parallel q}\ (\mathrm{par})$$

$$\frac{p \rightarrow_{\bar{a}} p' \quad q \rightarrow_a q'}{p \parallel q \rightarrow_\tau p' \parallel q'}\ (\mathrm{synch})$$

$$\frac{p \rightarrow_\alpha p' \quad \alpha \neq a, \bar{a}}{(a)p \rightarrow_\alpha (a)p'}\ (\mathrm{res}) \qquad \frac{p \equiv p' \rightarrow_\alpha q' \equiv q}{p \rightarrow_\alpha q}\ (\mathrm{equiv})$$

$$\mathrm{D}(\tilde{x}) \equiv p \text{ if } (D(\tilde{x}) := p) \in \Delta$$

Mathematical tools

$K \subseteq A$: *observable* actions (ex. $K = A \backslash \{\tau\}$)
$K^c := A \backslash K$.

A relation R is a **simulation** if $p\ R\ q$ implies:

$$\begin{tikzcd} p \arrow[r, dash, "R"] \arrow[d, "a \in K"'] & q \arrow[d, dotted, "w \in (K^c)^* a (K^c)^*", "*"'] \\ p' \arrow[r, dotted, dash, "R"] & q' \end{tikzcd} \qquad \begin{tikzcd} p \arrow[r, dash, "R"] \arrow[d, "a \in K^c"'] & q \arrow[d, dotted, "w \in (K^c)^*", "*"'] \\ p' \arrow[r, dotted, dash, "R"] & q' \end{tikzcd}$$

It is also a **bisimulation** if symmetric.

A solution . . .

$$\begin{array}{lcl} \mathrm{NODE}_i & \overset{def}{=} & \tau.(\mathrm{BUILD}_i^{\delta(i)} \mid \mathrm{WAIT}_{i\star}^{\delta(i)}) \\ & + & \sum_{j \in V} r_{ij}.(\mathrm{BUILD}_i^{\delta(i)-1} \mid \mathrm{WAIT}_{ij}^{\delta(i)-1}) \end{array}$$

A solution . . .

$$\begin{aligned}
\mathrm{NODE}_i \quad & \stackrel{def}{=} \quad \tau.(\mathrm{BUILD}_i^{\delta(i)} \mid \mathrm{WAIT}_{i\star}^{\delta(i)}) \\
& + \quad \sum_{j \in V} r_{ij}.(\mathrm{BUILD}_i^{\delta(i)-1} \mid \mathrm{WAIT}_{ij}^{\delta(i)-1})
\end{aligned}$$

$$\begin{aligned}
\mathrm{BUILD}_i^{n+1} \quad & \stackrel{def}{=} \quad \sum_{j \in V} \bar{r}_{ij}.\mathrm{BUILD}_i^{n} \\
\mathrm{BUILD}_i^{0} \quad & \stackrel{def}{=} \quad 0
\end{aligned}$$

A solution . . .

$$\begin{aligned}
\mathrm{NODE}_i &\stackrel{def}{=} \tau.(\mathrm{BUILD}_i^{\delta(i)} \mid \mathrm{WAIT}_{i\star}^{\delta(i)}) \\
&+ \sum_{j\in V} r_{ij}.(\mathrm{BUILD}_i^{\delta(i)-1} \mid \mathrm{WAIT}_{ij}^{\delta(i)-1})
\end{aligned}$$

$$\begin{aligned}
\mathrm{BUILD}_i^{n+1} &\stackrel{def}{=} \sum_{j\in V} \bar{r}_{ij}.\mathrm{BUILD}_i^{n} \\
\mathrm{BUILD}_i^{0} &\stackrel{def}{=} 0
\end{aligned}$$

$$\begin{aligned}
\mathrm{WAIT}_{i\alpha}^{n+1} &\stackrel{def}{=} w_i.\mathrm{WAIT}_{i\alpha}^{n} \\
\mathrm{WAIT}_{ij}^{0} &\stackrel{def}{=} \bar{w}_j.\uparrow_j^i \\
\mathrm{WAIT}_{i\star}^{0} &\stackrel{def}{=} \underline{ok_i}.\uparrow_\star^i
\end{aligned}$$

. . . with deadlocks

If $\delta(a) = 2$ and $\delta(b) = \delta(c) = 1$

$$
\begin{array}{rcl}
\mathrm{NODE}_a \mid \mathrm{NODE}_b \mid \mathrm{NODE}_c & \rightarrow & \mathrm{BUILD}^2_a \mid \mathrm{WAIT}^2_{a\star} \mid \mathrm{NODE}_b \mid \mathrm{NODE}_c \\
& \rightarrow^\star & \mathrm{WAIT}^2_{a\star} \mid \mathrm{WAIT}^0_{ba} \mid \mathrm{WAIT}^0_{ca} \\
& \equiv & w_a.w_a.\underline{ok_a}.\uparrow^a_\star \mid \bar{w}_a.\uparrow^b_a \mid \bar{w}_a.\uparrow^c_a \\
& \rightarrow^\star & \underline{ok_a}.\uparrow^a_\star \mid \uparrow^b_a \mid \uparrow^c_a \\
& \rightarrow_{\underline{ok_a}} & \uparrow^a_\star \mid \uparrow^b_a \mid \uparrow^c_a
\end{array}
$$

. . . with deadlocks

If $\delta(a) = 2$ and $\delta(b) = \delta(c) = 1$

$$\begin{array}{lll} \mathrm{NODE}_a \mid \mathrm{NODE}_b \mid \mathrm{NODE}_c & \rightarrow & \mathrm{BUILD}^2_a \mid \mathrm{WAIT}^2_{a\star} \mid \mathrm{NODE}_b \mid \mathrm{NODE}_c \\ & \rightarrow^\star & \mathrm{WAIT}^2_{a\star} \mid \mathrm{WAIT}^0_{ba} \mid \mathrm{WAIT}^0_{ca} \\ & \equiv & w_a.w_a.\underline{ok_a}.\uparrow^a_\star \mid \overline{w_a}.\uparrow^b_a \mid \overline{w_a}.\uparrow^c_a \\ & \rightarrow^\star & \underline{ok_a}.\uparrow^a_\star \mid \uparrow^b_a \mid \uparrow^c_a \\ & \rightarrow_{\underline{ok_a}} & \uparrow^a_\star \mid \uparrow^b_a \mid \uparrow^c_a \end{array}$$

If $\delta(a) = \delta(b) = 1$ and $\delta(c) = 3$

$$\begin{array}{lll} \mathrm{NODE}_a \mid \mathrm{NODE}_b \mid \mathrm{NODE}_c & \rightarrow & \mathrm{BUILD}^1_a \mid \mathrm{WAIT}^1_{a\star} \mid \mathrm{NODE}_b \mid \mathrm{NODE}_c \\ & \rightarrow & \mathrm{WAIT}^1_{a\star} \mid \mathrm{NODE}_b \mid \mathrm{BUILD}^2_c \mid \mathrm{WAIT}^2_{ca} \\ & \rightarrow & \mathrm{WAIT}^1_{a\star} \mid \mathrm{WAIT}^0_{bc} \mid \mathrm{BUILD}^1_c \mid \mathrm{WAIT}^2_{ca} \\ & \equiv & \mathrm{WAIT}^1_{a\star} \mid \overline{w_c}.\uparrow^b_c \mid \mathrm{BUILD}^1_c \mid w_c.w_c.\overline{w_a}.\uparrow^c_a \\ & \rightarrow & \mathrm{WAIT}^1_{a\star} \mid \uparrow^b_c \mid \mathrm{BUILD}^1_c \mid w_c.\overline{w_a}.\uparrow^c_a \end{array}$$

Not enough ?

The implementation is not bisimilar to the specification
But has some good properties:

- What is assembled is allowed
- It may find a way to simulate the specification

RCCS : syntax

$$p, q \quad ::= \quad (p \parallel q) \mid \sum_i \alpha_i.p_i \mid (a)p \mid D(\tilde{x}) \mid 0 \qquad \text{CCS}$$

$$\begin{array}{lll} r, s & ::= & m \triangleright p \\ & & \mid (r \parallel s) \\ & & \mid (a)r \end{array} \qquad \begin{array}{l} \text{Threads} \\ \text{Parallel} \\ \text{Restriction} \end{array}$$

$$\begin{array}{lll} m & ::= & \langle \theta, a, p \rangle \cdot m \\ & & \mid \ \langle\!\lbrace \theta \rbrace\!\rangle \cdot m \\ & & \mid \ \langle 1 \rangle \cdot m \ \mid \ \langle 2 \rangle \cdot m \\ & & \mid \ \langle \rangle \end{array} \qquad \begin{array}{l} \text{Synch} \\ \text{Commit} \\ \text{Fork address} \\ \text{Empty} \end{array}$$

RCCS : semantics

Transition:	$t = \langle r, \theta, \zeta, r' \rangle$	
Labels:	$\theta \in \mathcal{I}$	$\zeta := \alpha \mid \alpha^-$

If $\theta \notin \mathcal{I}(m)$:

$$m \triangleright \alpha.p + q \xrightarrow{\theta:\alpha} \langle \theta, \alpha, q \rangle \cdot m \triangleright p \quad \text{(act)}$$

$$\langle \theta, \alpha, q \rangle \cdot m \triangleright p \xrightarrow{\theta:\alpha^-} m \triangleright \alpha.p + q \quad (\text{act}^-)$$

$$m \triangleright \underline{\alpha}.p + q \xrightarrow{\theta:\underline{\alpha}} \langle\!| \theta |\!\rangle \cdot m \triangleright p \quad (\underline{\text{act}})$$

RCCS : semantics

Synchronisation rules are:

$$(\mathrm{com})\frac{r \xrightarrow{\theta:\bar{a}} r' \quad s \xrightarrow{\theta:a} s'}{r \parallel s \xrightarrow{\theta:\tau} r' \parallel s'} \qquad \frac{r \xrightarrow{\theta:\bar{a}^-} r' \quad s \xrightarrow{\theta:a^-} s'}{r \parallel s \xrightarrow{\theta:\tau^-} r' \parallel s'}(\mathrm{com}^-)$$

$$\frac{r \xrightarrow{\theta:\underline{\bar{a}}} r' \quad s \xrightarrow{\theta:\underline{a}} s'}{r \parallel s \xrightarrow{\theta:\underline{\tau}} r' \parallel s'}(\underline{\mathrm{com}})$$

RCCS : semantics

$$\begin{array}{lcl} m \triangleright (x)p & \equiv & (x)(m \triangleright p) \text{ if } x \notin m \\ m \triangleright (p \parallel q) & \equiv & (\langle 1 \rangle \cdot m \triangleright p) \parallel (\langle 2 \rangle \cdot m \triangleright q) \end{array}$$

Context rules are:

$$\frac{r \xrightarrow{\theta:\zeta} r' \quad \theta \notin \mathcal{I}(s)}{r \parallel s \xrightarrow{\theta:\zeta} r' \parallel s} \text{(par)} \qquad \frac{r \xrightarrow{\theta:\zeta} r' \quad \zeta \neq x, \bar{x}, x^-, \bar{x}^-}{(x)r \xrightarrow{\theta:\zeta} (x)r'} \text{(res)}$$

$$\frac{r \equiv r' \xrightarrow{\theta:\zeta} s' \equiv s}{r \xrightarrow{\theta:\zeta} s} \text{(equiv)}$$

Causal traces

A trace σ is said to be causal if:

1. there is only one irreversible action t
2. for all $\sigma' \sim \sigma$, σ' ends with t.

Causal traces

A trace σ is said to be causal if:

1. there is only one irreversible action t
2. for all $\sigma' \sim \sigma$, σ' ends with t.

$$a.\underline{b}.0 \mid c.0 \ \rightarrow_a \ \underline{b}.0 \mid c.0 \ \rightarrow_c \ \underline{b}.0 \ \rightarrow_{\underline{b}} \ 0$$

$$a.\underline{b}.0 \mid c.0 \ \rightarrow_a \ \underline{b}.0 \mid c.0 \ \rightarrow_{\underline{b}} \ c.0$$

$$a.\underline{b}.0 \mid \bar{a}.0 \ \rightarrow_\tau \ \underline{b}.0 \ \rightarrow_{\underline{b}} \ 0$$

Main theorem

Some definitions:

- K a set of underlined action in CCS
- $p_1 \rightarrow^{c}_{\underline{k}} p_2$ if there is a causal trace from p_1 to p_2 ending with $\underline{k} \in K$
- $\mathrm{CTS}(p) = (P, p, K, \rightarrow^{c})$: the causal transition system induced by p

Theorem

Let p be a CCS process and Φ the relation $\{(\underline{k}, \theta : \underline{k}) \mid \underline{k} \in K, \theta \in \mathcal{I}\}$, then $CTS(p) \approx_{\Phi} LTS(\langle\rangle \triangleright p)$.

On the implementation

What is not part of the theorem:

- mapping the trees into the CCS code:

$$[\![(a, \{t_1, \ldots, t_n\})]\!]_\alpha \quad = \quad \uparrow^a_\alpha \mid [\![t_1]\!]_a \mid \ldots \mid [\![t_n]\!]_a$$

- mapping the states of the LTS into the CCS code:

$$[\![N, \textstyle\sum_i t_i]\!] \quad = \quad \prod_{i \in N} \mathrm{NODE}_i \mid \prod_j [\![t_j]\!]_\star$$

Proposition

Let Φ be the relation $\left\{(t, \underline{ok_i}) \mid i \in V\right\}$.
The relation $\{(N, \sum_i t_i), [\![N, \sum_i t_i]\!]\}$ is a Φ-bisimulation between SPEC *and* $CTS([\![V]\!])$.

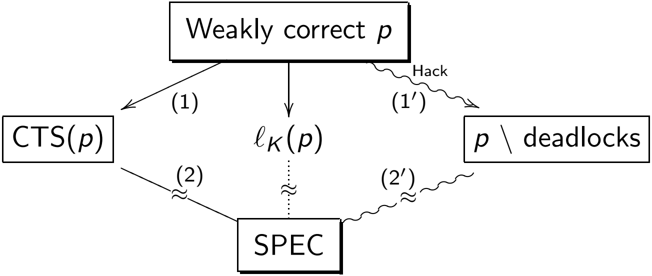
Proving correctness in RCCS

Bisimulation is 1. and 2. *weak correctness* is 1. and 2'.

1. **Simulation:** All transactions of the spec. can be performed in the implementation.
2. **Correctness:** All evolutions of the implementation lead to a state which is also in the spec.

2'. **No bad state:** All evolutions of the implementation that (causally) lead to a state must be in accordance with the specification.

Declarative Concurrent Programming



(1) Automatic: Ocaml tool (`Causal`)

(1') By hand. Difficulty depends on system's topology.

(2) Automatic: in most cases CTS(p) $\equiv$ SPEC.

(2') By hand beyond a certain size

Further works

Some drawbacks:

- General result may imply loss of efficiency when backtracking
- We strongly rely on the mapping $[\![\cdot]\!]$

Improvements:

- Having more refined labels (CCS with values)
- Dealing with graphs (Reversible π-calculus)
- Handling a stronger property on the correctness (stochastic behaviour)