Tools for CSP

Markus Roggenbach, Swansea (Wales)
cooperation with Y Isobe, IAST Tsukuba (Japan)

Bamberg, April 2012

The process algebra CSP

- Established formalism to describe concurrent systems.
- Ongoing research regarding foundations.
- Applications in industry include
  Train Controllers, Avionics, Security Protocols.

Hoare. Communicating Sequential Processes. Prentice Hall, 1985.

Ryan et al. Security Protocols. Addison Wesley 2001.

Abdallah, Jones, Sanders (eds). CSP: The First 25 Years. Springer 2005.

Roscoe. Understanding Concurrent Systems. Springer, 2010.

Overview

A sample system

Modelling, Simulation & Model-Checking

Theorem-Proving on CSP specifications

Validating CSP

A glimpse on Timed CSP

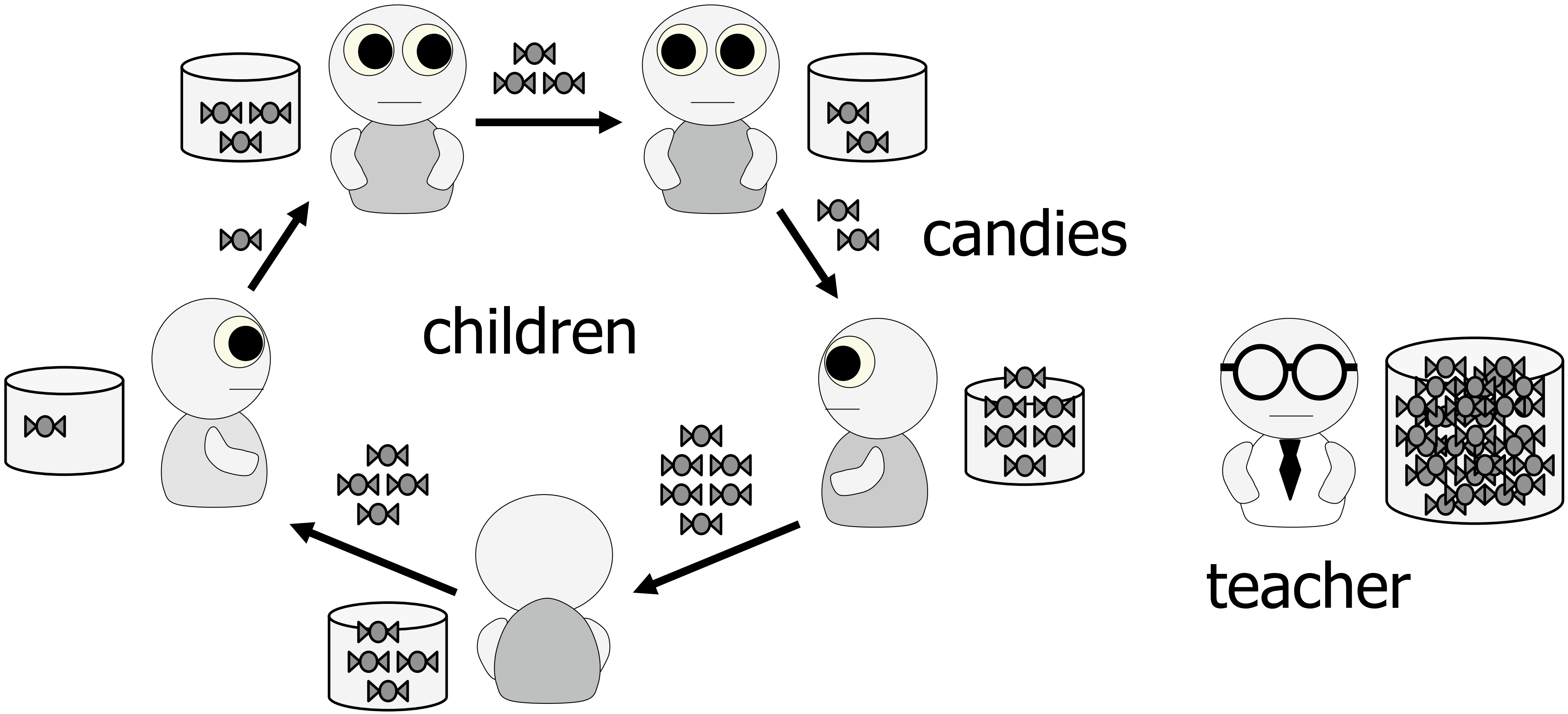
A sample system

The children & candy puzzle

There are n children sitting in a circle,
each with an even number of candies.

The following two steps are repeated indefinitely:

- every child passes half of their candies to the child on their left;
- any child who ends up with an odd number of candies is given another candy by the teacher.


candies
children
teacher

Some natural questions on the system

- Will the teacher keep handing out more and more candies?
- Will an unequal distribution of candies eventually become an equal one?

Reasoning about the system

Claim:

The maximum number of candies
held by a single child
never increases.

Proof

Let c be one of the children.

Proof

Let c be one of the children.

State before entering the loop:

Let $2M$ be the maximum number of candies a child holds.

- Child c holds $2K \leq 2M$ candies.

Proof

Let c be one of the children.

State before entering the loop:

Let $2M$ be the maximum number of candies a child holds.

- Child c holds $2K \leq 2M$ candies.

In the first step, child c

- gives away K candies,
- keeps $K \leq M$ candies, and
- receives $L \leq M$ candies.

Thus, afterwards child c holds $K + L$ candies.

Proof (continued)

Filling up from the teacher yields:

- if $K = L = M$ or $K = L = M - 1$:
  $K + L$ is even, no fill up. $\leadsto$ Child c holds $\leq 2M$ candies.

Proof (continued)

Filling up from the teacher yields:

- if $K = L = M$ or $K = L = M - 1$:
  $K + L$ is even, no fill up. $\leadsto$ Child c holds $\leq 2M$ candies.
- if $K = M - 1,\ L = M$
  or $K = M,\ L = M - 1$:
  $K + L$ is odd, fill up by 1. $\leadsto$ Child c holds $2M$ candies.

Proof (continued)

Filling up from the teacher yields:

- if $K = L = M$ or $K = L = M - 1$:
  $K + L$ is even, no fill up. $\leadsto$ Child c holds $\leq 2M$ candies.
- if $K = M - 1,\ L = M$
  or $K = M,\ L = M - 1$:
  $K + L$ is odd, fill up by 1. $\leadsto$ Child c holds $2M$ candies.
- if $K, L < M - 1$:
  filling up by at most one candy
  $\leadsto$ Child c holds less than $2M$ candies.

Proof (continued)

Filling up from the teacher yields:

- if $K = L = M$ or $K = L = M - 1$:
  $K + L$ is even, no fill up. $\leadsto$ Child c holds $\leq 2M$ candies.
- if $K = M - 1,\ L = M$
  or $K = M,\ L = M - 1$:
  $K + L$ is odd, fill up by 1. $\leadsto$ Child c holds $2M$ candies.
- if $K, L < M - 1$:
  filling up by at most one candy
  $\leadsto$ Child c holds less than $2M$ candies.

In all three cases child c holds less or equal to $2M$ candies.

Further insights into the puzzle

- The maximum number of candies held by a single child never increases.

  Consequence:

  The teacher must eventually stop handing out candies.

Further insights into the puzzle

- The maximum number of candies held by a single child never increases.

  Consequence:

  The teacher must eventually stop handing out candies.

- Eventually,

  all children will hold the same number of candies.

Reflection

- The proof arguments are on arithmetic only.

Reflection

- The proof arguments are on arithmetic only.
- Concurrency does not appear at all.

Reflection

- The proof arguments are on arithmetic only.
- Concurrency does not appear at all.

Report on what happens if one stops here:

Peleska et al. Formal Methods for the International Space Station. 1999.

Modelling, Simulation & Model-Checking

Asynchronous model of the puzzle in CSP

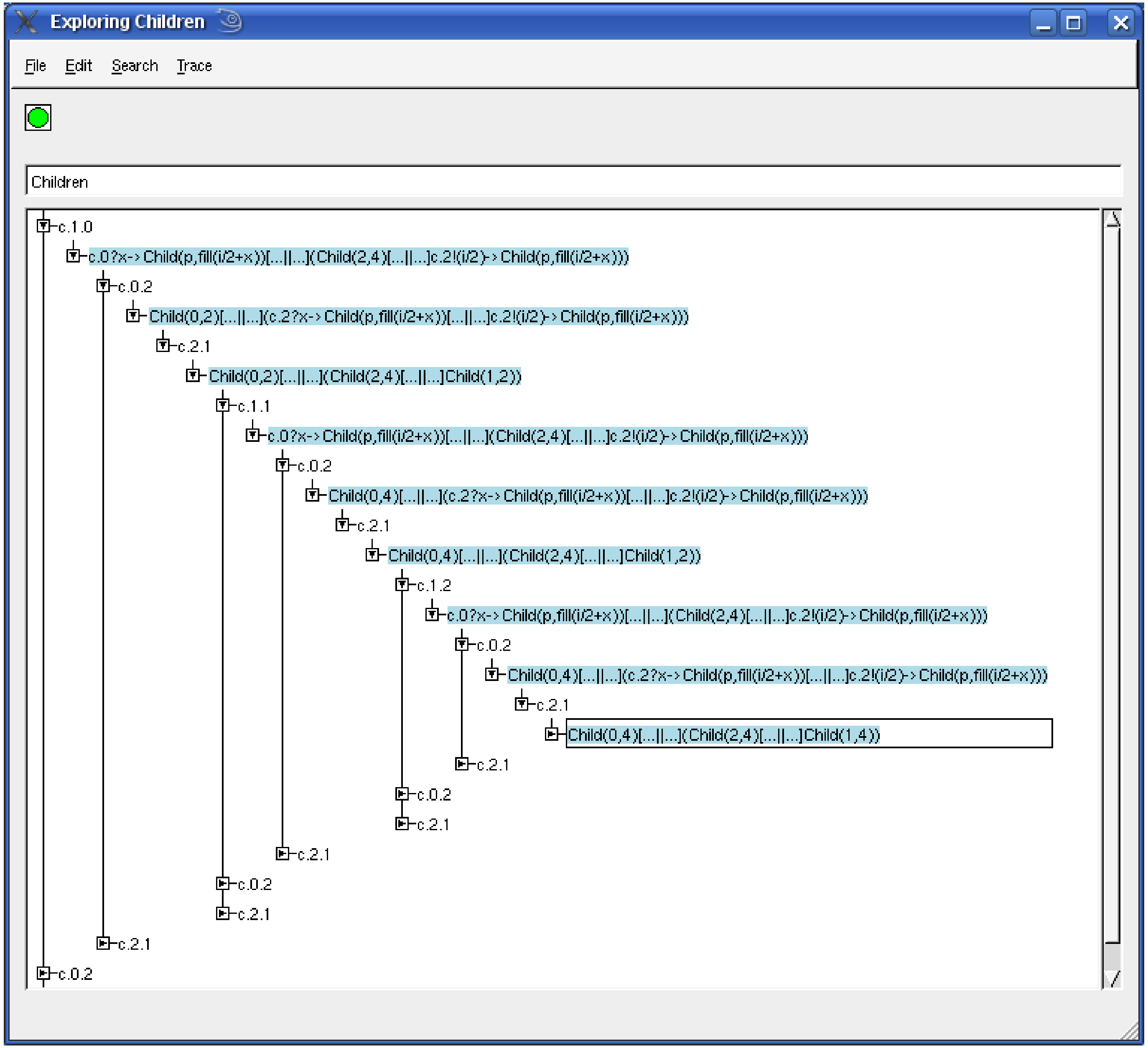
```
fill(n) = if (n%2==0) then n else n+1
channel c: {0..2}.{0..2}

-- One child process:
Child(p,i) =
   (   c.(p+1)%3 ! (i/2)
    -> c.p       ? x       -> Child(p, fill(i/2+x)))
[] (   c.p       ? x
    -> c.(p+1)%3 ! (i/2) -> Child(p, fill(i/2+x)))

-- 3 Children, holding 2*p candies each:
Children = ||p:{0..2}@[ {| c.p, c.(p+1)%3 |} ] Child(p,2*p)
```

Simulation with ProBE

$\boxed{\text{Simulate example of 3 children with 0,2,4 candies}}$

ProBE: Process Behaviour Explorer

Free download from Formal Systems


Exploring Children
File Edit Search Trace
Children
c.1.0
c.0?x-> Child(p,fill(i/2+x))[...||...](Child(2,4)[...||...]c.2!(i/2)-> Child(p,fill(i/2+x)))
c.0.2
Child(0,2)[...||...](c.2?x-> Child(p,fill(i/2+x))[...||...]c.2!(i/2)-> Child(p,fill(i/2+x)))
c.2.1
Child(0,2)[...||...](Child(2,4)[...||...]Child(1,2))
c.1.1
c.0?x-> Child(p,fill(i/2+x))[...||...](Child(2,4)[...||...]c.2!(i/2)-> Child(p,fill(i/2+x)))
c.0.2
Child(0,4)[...||...](c.2?x-> Child(p,fill(i/2+x))[...||...]c.2!(i/2)-> Child(p,fill(i/2+x)))
c.2.1
Child(0,4)[...||...](Child(2,4)[...||...]Child(1,2))
c.1.2
c.0?x-> Child(p,fill(i/2+x))[...||...](Child(2,4)[...||...]c.2!(i/2)-> Child(p,fill(i/2+x)))
c.0.2
Child(0,4)[...||...](c.2?x-> Child(p,fill(i/2+x))[...||...]c.2!(i/2)-> Child(p,fill(i/2+x)))
c.2.1
Child(0,4)[...||...](Child(2,4)[...||...]Child(1,4))
c.2.1
c.0.2
c.2.1
c.2.1
c.0.2
c.2.1
c.2.1
c.0.2

Modelling self-stabilization

```
StableAfter (n) =
  if n>0 then  c.0 ? x -> StableAfter (n-1)
            [] c.1 ? x -> StableAfter (n-1)
            [] c.2 ? x -> StableAfter (n-1)
  else Stable

Stable =       c.0 ! 2 -> Stable
            [] c.1 ! 2 -> Stable
            [] c.2 ! 2 -> Stable
```

Model-checking

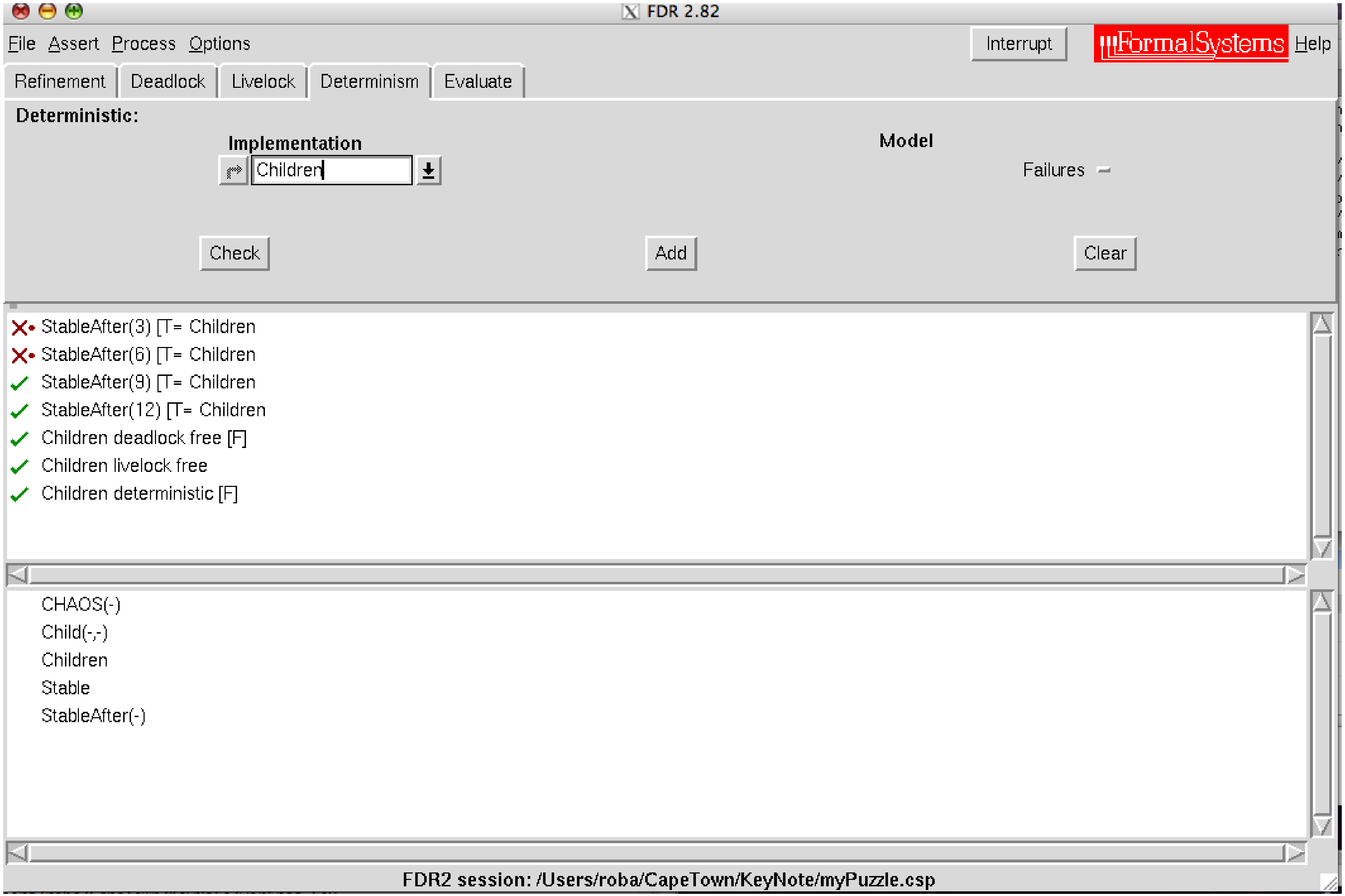
Is our model

1. self-stabilized after 3, 6, 9, or 12 steps?
2. deadlock-free?
3. lifelock-free?
4. deterministic?

Check example of 3 Children with 0,2,4 candies

FDR: Failures Divergences Refinement

Free academic licence available from Formal Systems


FDR 2.82
File Assert Process Options
Interrupt
FormalSystems
Help
Refinement Deadlock Livelock Determinism Evaluate
Deterministic:
Implementation
Children
Model
Failures
Check
Add
Clear
StableAfter(3) [T= Children
StableAfter(6) [T= Children
StableAfter(9) [T= Children
StableAfter(12) [T= Children
Children deadlock free [F]
Children livelock free
Children deterministic [F]
CHAOS(-)
Child(-,-)
Children
Stable
StableAfter(-)
FDR2 session: /Users/roba/CapeTown/KeyNote/myPuzzle.csp

Reflection

- In CSP, we made a concurrent machine model precise.
- In ProBE, we can simulate a single instance of our puzzle.
- In FDR, we can verify a single instance of our puzzle.

The mathematical proof is stronger:

for all numbers of childeren and
for all initial distributions of candy
eventually we will reach an even distribution.

Theorem-Proving on CSP specifications

A ‘new’ trend in process algebra

CSP

HOL-CSP (Tej/Wolff 1997)
Schneider/Dutertre (2001)
CSP-Prover (2005)
Wei/Heather (2005)
Kammüller (2007)
Göthel/Glesner(2010)

CCS

Nesi (1992)

μCRL/ACP

van de Pol (2001)
Badban et al (2005)

π-calculus

Röckl/Hirschkoff (2003)
Bengtson/Parrow (2007)
Kahsai/Miculan (2008)

Children's puzzle in CSP-Prover

For all lists s of even numbers with $length(s) \geq 2$ holds:

Arithmetic property:

$$\exists\, n.\ \max(circNext^{(n)}(s)) = \min(circNext^{(n)}(s)).$$

where $circNext : \mathbf{N}^* \rightarrow \mathbf{N}^*$.

Children's puzzle in CSP-Prover

For all lists s of even numbers with $length(s) \geq 2$ holds:

Arithmetic property:

$$\exists\, n.\ \max(circNext^{(n)}(s)) = \min(circNext^{(n)}(s)).$$

where $circNext : \mathbf{N}^* \to \mathbf{N}^*$.

Process property:

$$\texttt{EventuallyStable}(s) \sqsubseteq_{\mathcal{F}} \texttt{Children}(s)$$

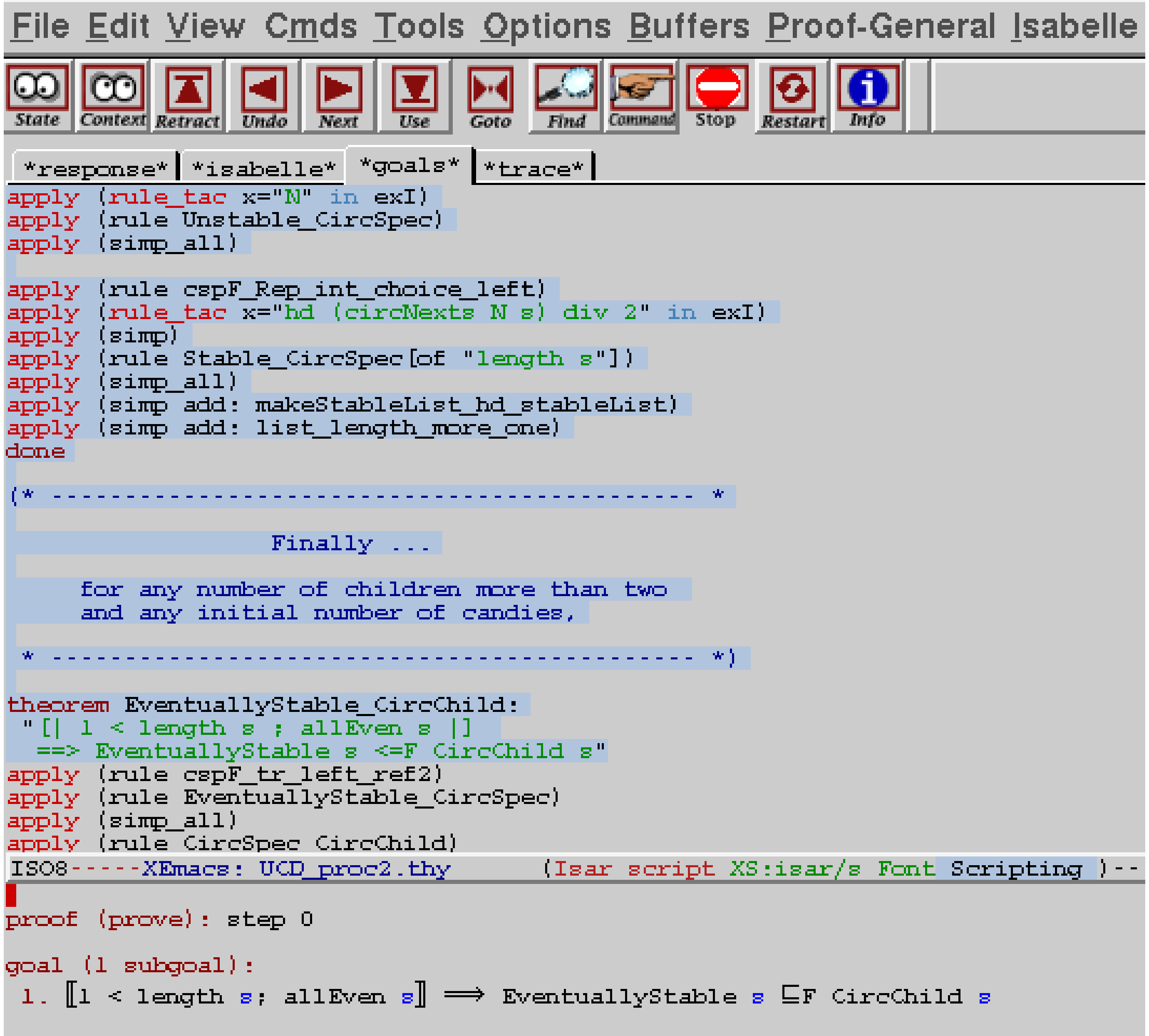
where $\texttt{EventuallyStable}(s) = c.0!(hd(s)/2) \to \texttt{EventuallyStable}(circNext(s))$.

Proof with CSP-Prover

Prove stabilization for all instances

CSP-Prover: interactive theorem prover

Free download from CSP-Prover website.


File Edit View Cmds Tools Options Buffers Proof-General Isabelle
State Context Retract Undo Next Use Goto Find Command Stop Restart Info
response *isabelle* *goals* *trace*
apply (rule_tac x="N" in exI)
apply (rule Unstable_CircSpec)
apply (simp_all)
apply (rule cspF_Rep_int_choice_left)
apply (rule_tac x="hd (circNexts N s) div 2" in exI)
apply (simp)
apply (rule Stable_CircSpec[of "length s"])
apply (simp_all)
apply (simp add: makeStableList_hd_stableList)
apply (simp add: list_length_more_one)
done
(* -- *
Finally ...
for any number of children more than two
and any initial number of candies,
* -- *)
theorem EventuallyStable_CircChild:
"[| 1 < length s ; allEven s |]
==> EventuallyStable s <=F CircChild s"
apply (rule cspF_tr_left_ref2)
apply (rule EventuallyStable_CircSpec)
apply (simp_all)
apply (rule CircSpec CircChild)
ISO8-----XEmacs: UCD_proc2.thy (Isar script XS:isar/s Font Scripting)--
proof (prove): step 0
goal (1 subgoal):
1. ⟦1 < length s; allEven s⟧ ⟹ EventuallyStable s ⊑F CircChild s

Reflection

- In CSP-Prover, we can verify whole classes of parametrized systems.

Concrete for the Children's puzzle:

The puzzle will stabilize for

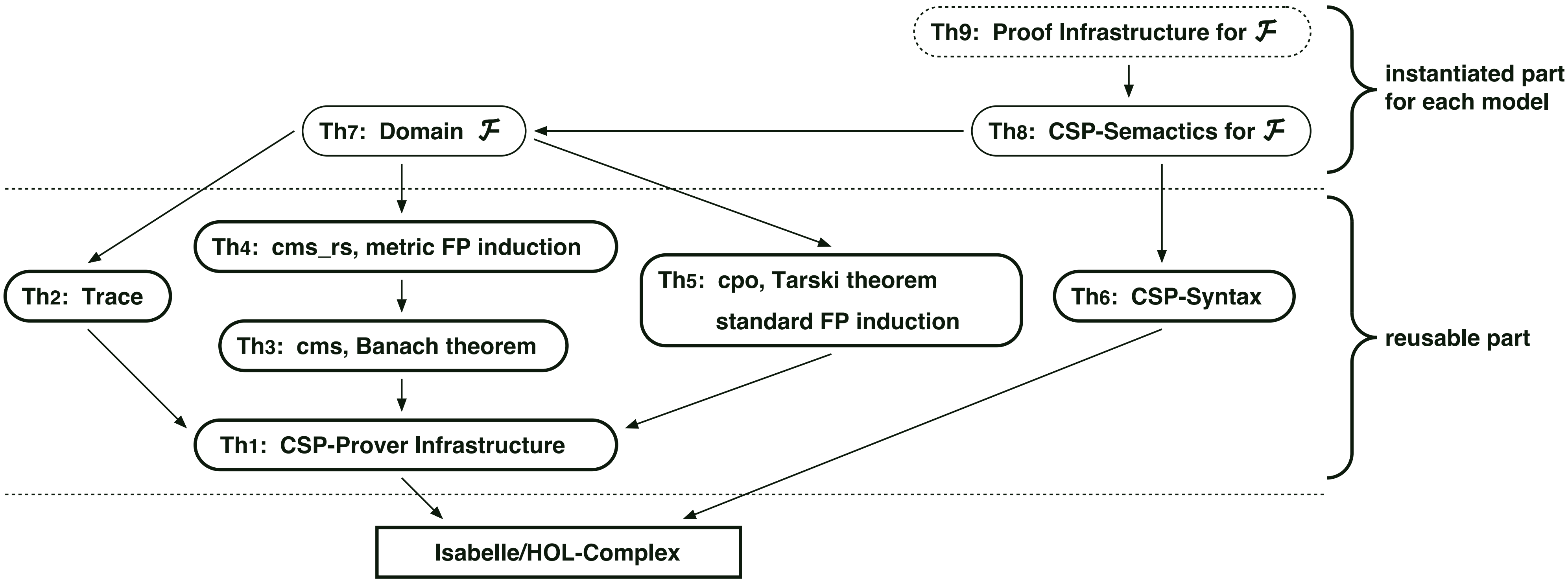
- all numbers $n \geq 2$ of children and for
- all initial candy distributions.

Case studies in CSP Theorem Proving

- Buffer examples (Tej/Wolff)
- Dining Mathematicians
- Dining Philosophers (Wei/Heather)
- Children's puzzle
- Systolic algorithms
- Needham-Schroeder protocol (Schneider/Dutertre)
- Electronic payment system
- . . .

Validating CSP

CSP-Prover: A deep embedding of CSP



Implemented CSP models in CSP-Prover: $\mathcal{T}, \mathcal{F}, \mathcal{R}$.

Correction of a step law from Roscoe '98:

$$P \triangleright Q := (P \sqcap Stop) \Box Q$$

Roscoe's version

$$\begin{aligned} & (P \triangleright P') \,|[\, X \,]|\, (Q \triangleright Q') \\ = \quad & (P \,|[\, X \,]|\, Q) \triangleright ((P' \,|[\, X \,]|\, (Q \triangleright Q')) \sqcap ((P \triangleright P') \,|[\, X \,]|\, Q')) \end{aligned}$$

Correction of a step law from Roscoe '98:

$P \triangleright Q := (P \sqcap Stop) \,\Box\, Q$

Roscoe's version

$$\begin{array}{rl} & (P \triangleright P') \,|[\, X \,]|\, (Q \triangleright Q') \\ = & (P \,|[\, X \,]|\, Q) \triangleright ((P' \,|[\, X \,]|\, (Q \triangleright Q')) \sqcap ((P \triangleright P') \,|[\, X \,]|\, Q')) \end{array}$$

Correction

Let $P = (?x : A \to P'(x)) \triangleright P''$, $Q = (?x : B \to Q'(x)) \triangleright Q''$

$$\begin{array}{rcl} P \,|[\, X \,]|\, Q & = & (?x : ((X \cap A \cap B) \cup (A - X) \cup (B - X)) \to \\ & & \mathit{if}\ (x \in X)\ \mathit{then}\ (P'(x) \,|[\, X \,]|\, Q'(x)) \\ & & \mathit{else\ if}\ (x \in A \cap B)\ \mathit{then}\ ((P'(x) \,|[\, X \,]|\, Q) \sqcap (P \,|[\, X \,]|\, Q'(x))) \\ & & \mathit{else\ if}\ (x \in A)\ \mathit{then}\ (P'(x) \,|[\, X \,]|\, Q)\ \mathit{else}\ (P \,|[\, X \,]|\, Q'(x))) \\ & & \triangleright ((P'' \,|[\, X \,]|\, Q) \sqcap (P \,|[\, X \,]|\, Q'')) \end{array}$$

Bill Roscoe's reaction

$>$ my colleague Yoshinao Isobe (AIST, Japan) and I found
$>$ counter examples to the step laws for . . .

You are right about them...

I think that, implicitly, it demonstrates that, soon, presentations of similar models and axiom schemes will only be "complete" once they have been accompanied by similar mechanised theorem proving.

A mistake in the semantics of R

The law

$$Stop =_{\mathcal{R}} \; ?\, x : \emptyset \rightarrow P(x)$$

fails in the original setting (Roscoe 2007, unpublished draft):

- $deadlocks(Stop) = \{\langle\rangle\}$.
- $deadlocks(?x : \emptyset \rightarrow P) = \{\}$.

In the publication (to appear): corrected deadlock clause.

Complete axiomatic semantics for CSP

Best known result:

Finitely non-deterministic CSP over a finite alphabet
(Roscoe 1998, improving Brookes 1983)

Complete axiomatic semantics for CSP

Best known result:

Finitely non-deterministic CSP over a finite alphabet
(Roscoe 1998, improving Brookes 1983)

Concur'06: Axiom system $A_{\mathcal{F}}$ ($\sim$ 80 cond. eq.)

$$A_{\mathcal{F}} \vdash P = Q \Leftrightarrow [\![P]\!]_{\mathcal{F}} = [\![Q]\!]_{\mathcal{F}}$$

Relative completeness for CSP over an arbitrary alphabet;
oracles for set theory & natural numbers
(side conditions require theorems on sets and naturals)

Reflection

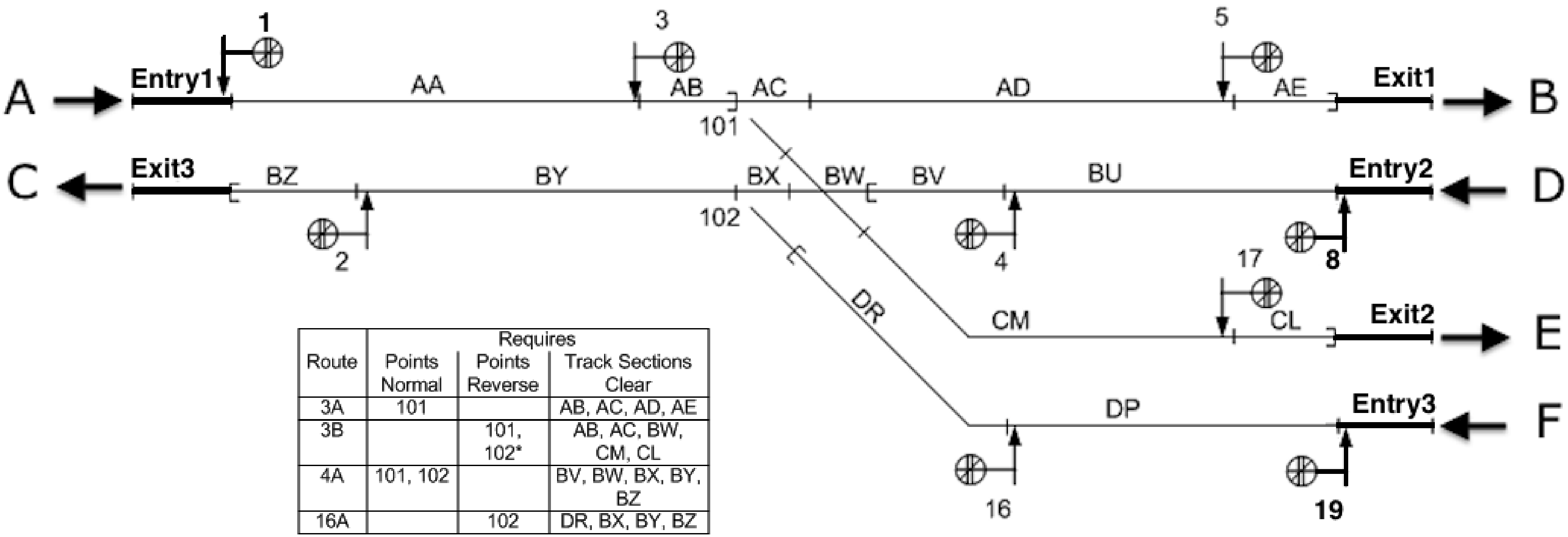
Thanks to its deep encoding,
CSP-Prover allows to reflect on CSP:

- Validation of CSP models.
- Verification of the algebraic laws of CSP.
- Proof of meta-theorems comparing semantical models.

A glimpse on Timed CSP

The SafeCap Project (Started in 2/2011)

Overcoming the railway capacity challenges
without undermining rail network safety.



Route	Requires		
	Points Normal	Points Reverse	Track Sections Clear
3A	101		AB, AC, AD, AE
3B		101, 102*	AB, AC, BW, CM, CL
4A	101, 102		BV, BW, BX, BY, BZ
16A		102	DR, BX, BY, BZ

* flank protection

The language Timed CSP

Time: Real number ≥ 0.

Newtonian time concept – single conceptual global clock.

The language Timed CSP

Time: Real number ≥ 0.

Newtonian time concept – single conceptual global clock.

Conservatively extends CSP by three primitives:

- $a@u \rightarrow P(u)$ – Time of an action.
- $P \triangleright^d Q$ – Timed timeout.
- $P \triangle_e Q$ – Timed interrupt.

The language Timed CSP

Time: Real number ≥ 0.

Newtonian time concept – single conceptual global clock.

Conservatively extends CSP by three primitives:

- $a @ u \to P(u)$ – Time of an action.
- $P \triangleright^{d} Q$ – Timed timeout.
- $P \triangle_{e} Q$ – Timed interrupt.

Syntactic sugar, e.g.

- $Wait\ d = Stop \triangleright^{d} Skip$.

Simulating Timed CSP

Theorem 1:

1. $P \xrightarrow{\tau}$ iff $time(P) = \{\}$.
2. $P \overset{\tau}{\not\to}$ iff $time(P) = \mathbb{R}_{>0} \vee time(P) = (0, lub(time(P))]$.

Theorem 2:

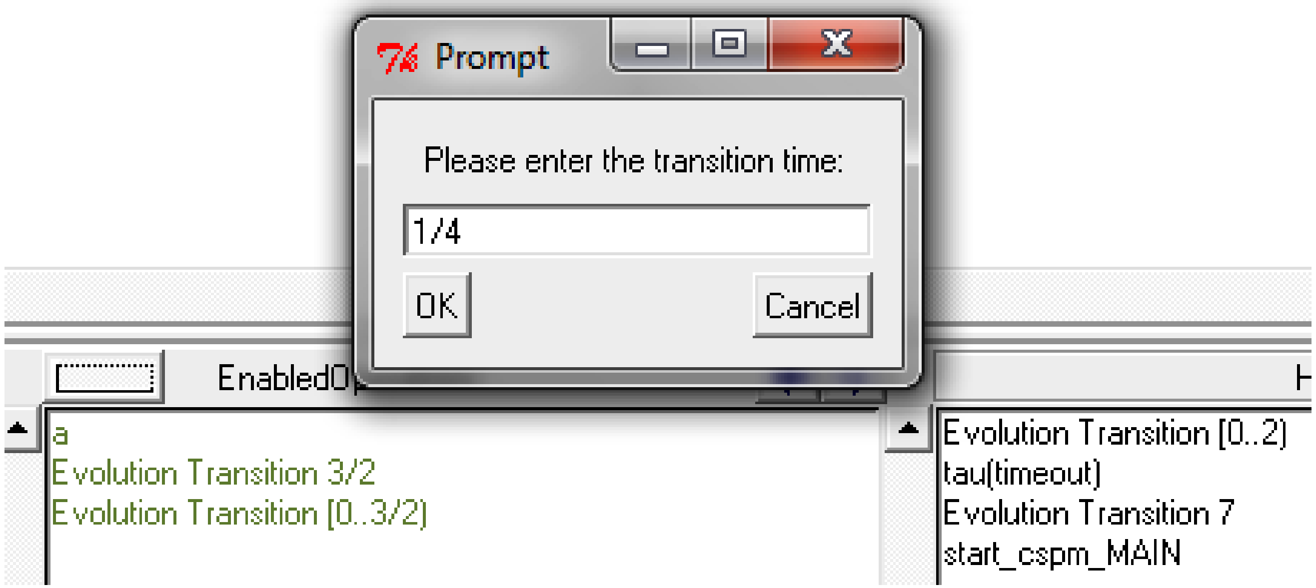
Rational processes are closed under action transitions and rational delays.

$P \xrightarrow{\tau}$ expands to $\exists\, P' : P \xrightarrow{\tau} P'$.

$P \overset{\tau}{\not\to}$ expands to $\neg(P \xrightarrow{\tau})$.

$time(P) = \{r \in \mathbb{R}_{>0} \mid \exists\, P' : P \overset{r}{\rightsquigarrow} P'\}$.

Timed CSP Simulator (AVoCS'11)



Extends Leuschel's CSP simulator in ProB.

Model-checking Timed CSP

1. Direct approach: Process Analysis Toolkit.
2. Translational approach:

$SPEC \sqsubseteq_{TT} IMP$ over Timed CSP

is equivalent to

$time(SPEC) \sqsubseteq_{T} time(IMP)$ over (untimed) CSP.

(under certain conditions)

First results in SafeCap

Changing control tables

- yields a capacity increase (one more train every 6 min)
- without compromising safety (collision freedom)

for single paths.

First results in SafeCap

Changing control tables

- yields a capacity increase (one more train every 6 min)
- without compromising safety (collision freedom)

for single paths.

Tools involved

- Timed CSP Simulator
- FDR (via translational approach)

First results in SafeCap

Changing control tables

- yields a capacity increase (one more train every 6 min)
- without compromising safety (collision freedom)

for single paths.

Tools involved

- Timed CSP Simulator
- FDR (via translational approach)

Analysis of the full example: requires theorem proving.

Conclusion

Summary

- CSP and Timed CSP have a rich set of supporting tools.
- Theorem Proving is an established trend.

Future work

- Extending CSP-Prover to a Timed CSP-Prover.
- Verification of the relations between CSP models.
- Structured specification and compositional proofs on CSP.

Including data specification

Motivation

. . . In order to make CSP useful in practice we have added quite a rich language of sub-process objects: . . . a functional programming language . . .

. . . any complete semantics of CSP . . . would need to give this sublanguage a semantics too.

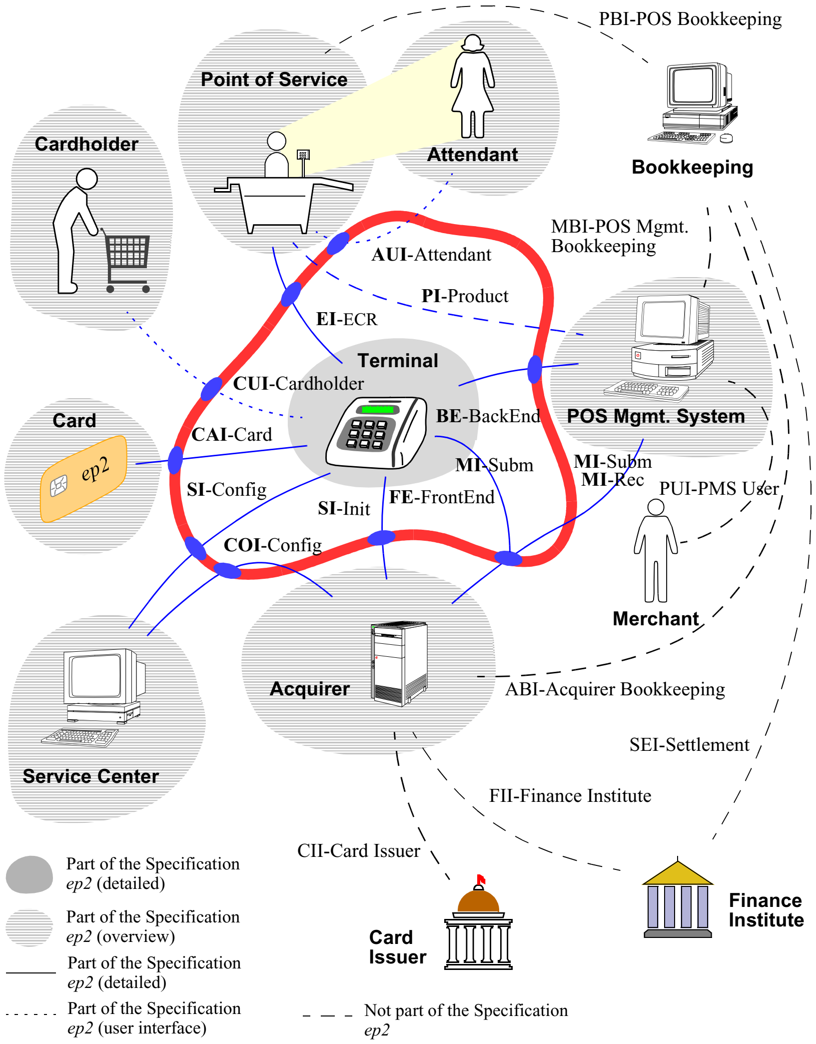
. . . this would take a lot of extra space . . .

[Roscoe: The Theory and Practice of Concurrency. 1998.]

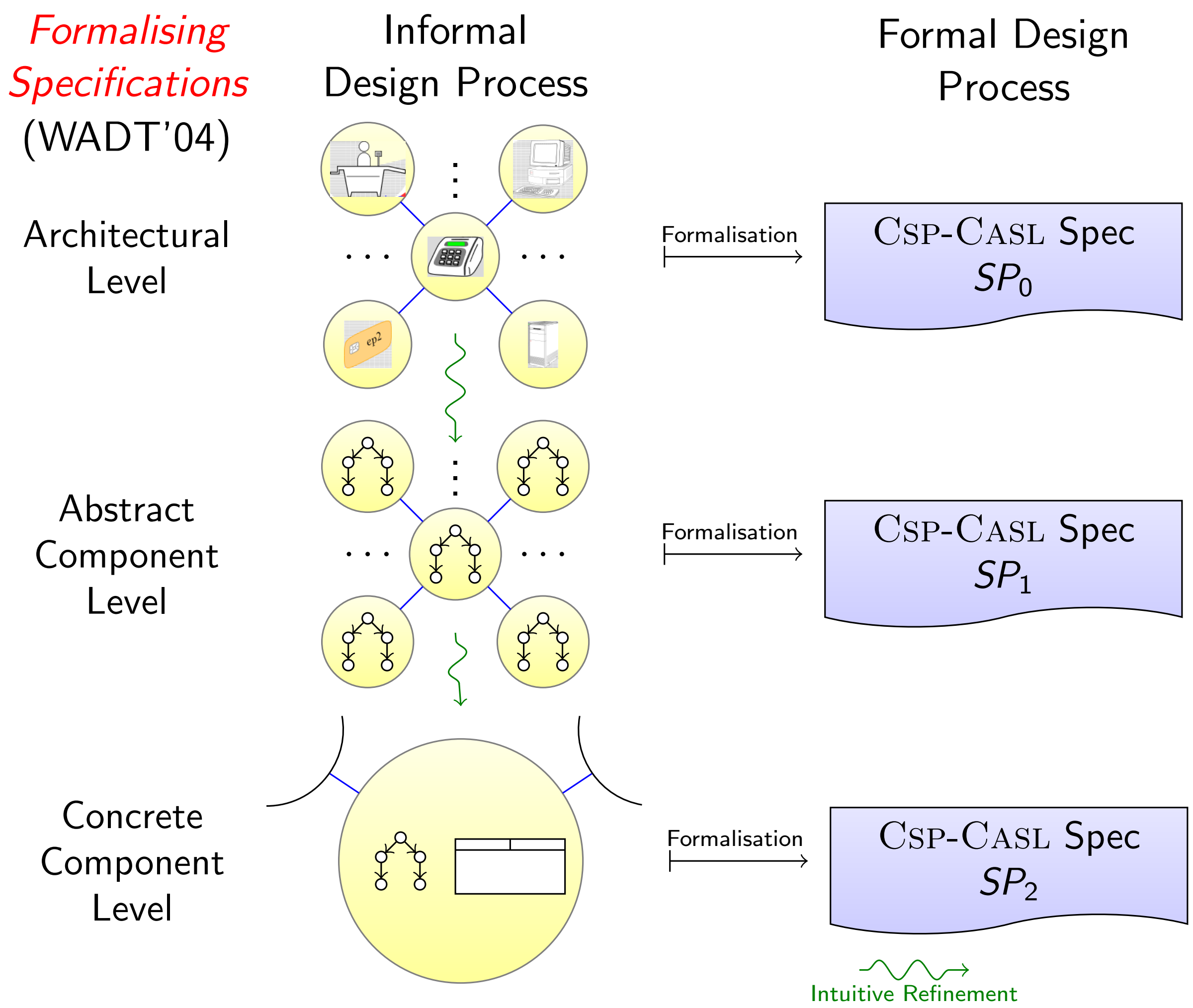
Many approaches for data & processes

- Lotos
- E-Lotos
- CASL-Charts
- CCS-CASL
- Circus (CSP + Z)
- CSP-CASL
- ...

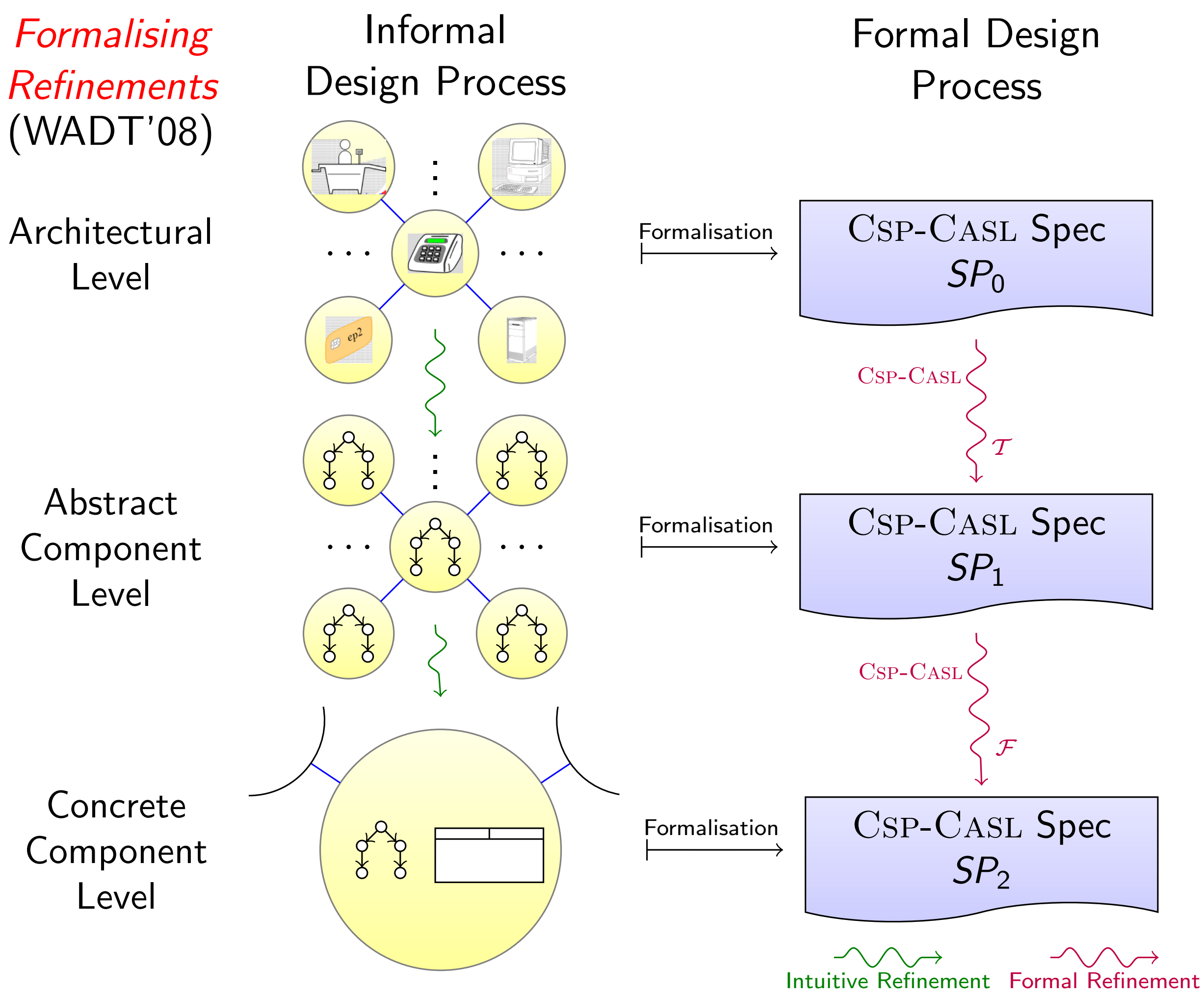
The electronic payment system EP2



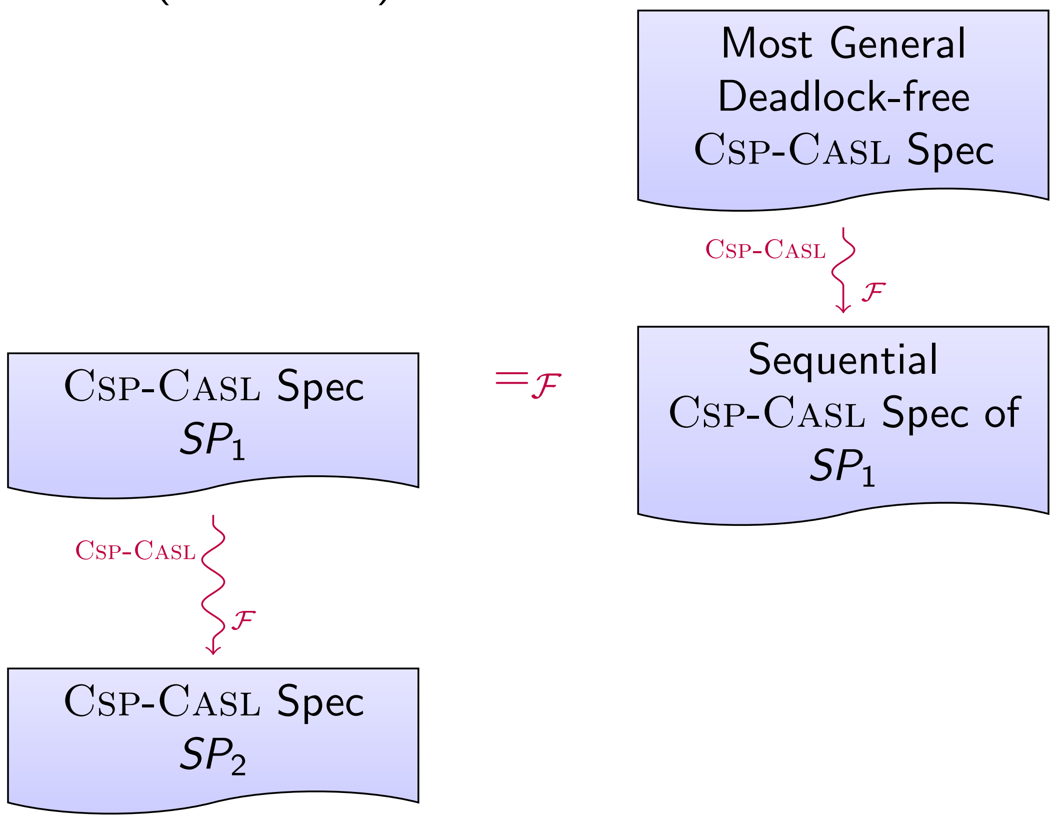
Modelling in CSP-CASL



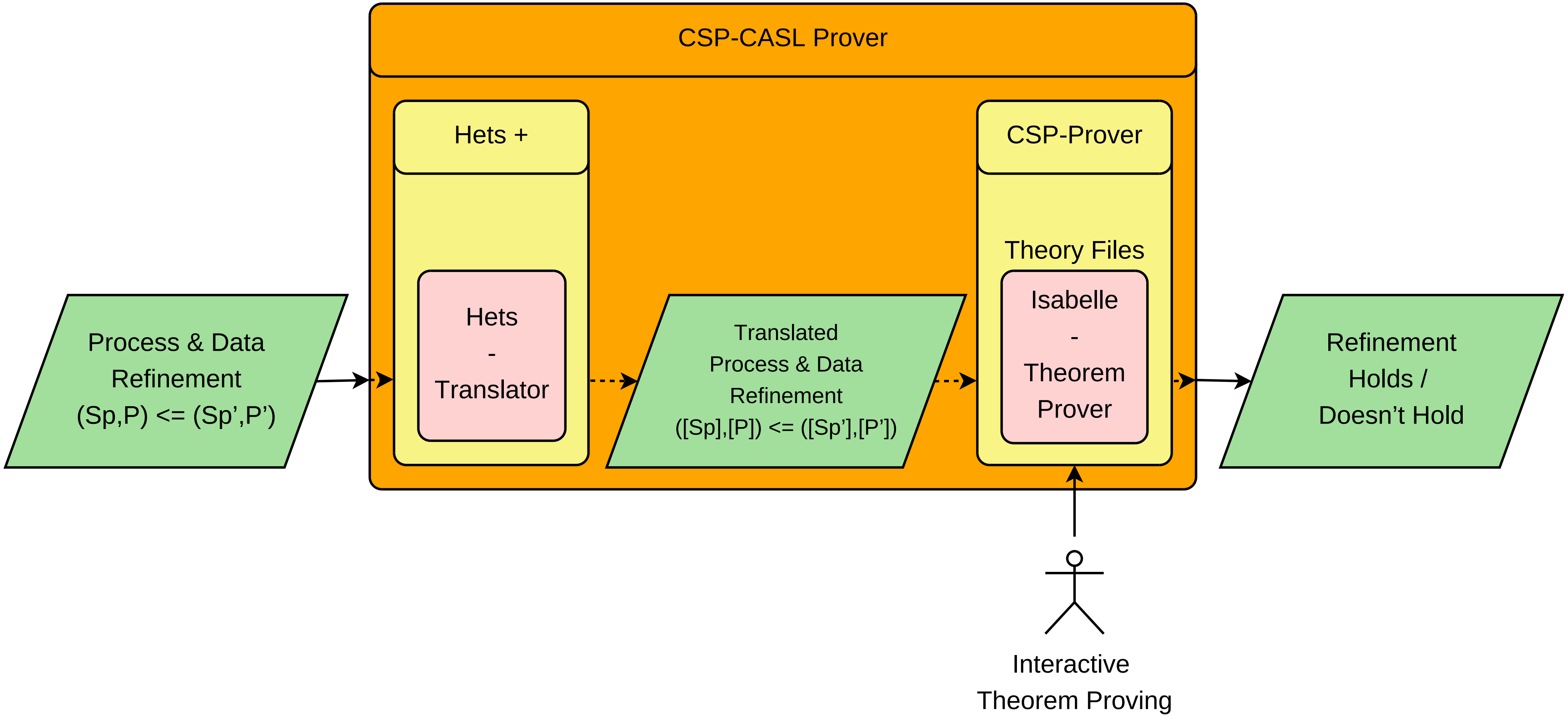
Proving refinements in CSP-CASL



Analysing systems: Deadlock-freedom
(WADT'08)



CSP-CASL-Prover



Reflection

- CSP has sub-process objects to represent data, for which
  - CSP_M offers a functional language.
  - CSP-CASL offers an algebraic specification language.
- Industrial systems require loose, abstract data types.
- CSP-CASL-Prover offers integrated theorem proving on processes and data.