Proof Principles of CSP – CSP-Prover in Practice

Yoshinao Isobe, AIST (Japan)
Markus Roggenbach, Swansea (Wales)

Bremen, August 2007

CSP = Communicating Sequential Processes

- Established formalism to describe concurrent systems.
- Ongoing research on foundations;
  Applications in industry, e.g.,
  Train Controllers, Avionics, Security Protocols.

Hoare: *Communicating Sequential Processes*, 1985

Roscoe: *The theory and practice of concurrency*, 1998 & 2005

Schneider et al: *Security Protocols: the CSP Approach*, 2001

Roscoe et al: *CSP, the first 25 years*, 2005

Outline

Theorem Proving for Process Algebra

CSP-Prover in Practice

Theorem Proving for Process Algebra

A ‘logistic problem’

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

The following step is repeated indefinitely:

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

And here is what is going to happen . . .

You might think that the teacher may keep handing out more and more candies indefinitely.

And here is what is going to happen . . .

You might think that the teacher may keep handing out more and more candies indefinitely.

However, this is not true.

And here is what is going to happen . . .

You might think that the teacher may keep handing out more and more candies indefinitely.

However, this is not true.

Eventually

- the teacher will stop handing out candies and
- every child will hold the same number of candies.

How to achieve such a result?

Model the system, e.g., in CSP and analyse it . . .

How to achieve such a result?

Model the system, e.g., in CSP and analyse it . . .

in the very beginning

- paper & pen only

How to achieve such a result?

Model the system, e.g., in CSP and analyse it . . .

in the very beginning

- paper & pen only

in the past

- plus Model Checking

How to achieve such a result?

Model the system, e.g., in CSP and analyse it . . .

in the very beginning

- paper & pen only

in the past

- plus Model Checking

currently

- plus Model Checking complemented by Theorem Proving

How to achieve such a result?

Model the system, e.g., in CSP and analyse it . . .

in the very beginning

- paper & pen only

in the past

- plus Model Checking

currently

- plus Model Checking complemented by Theorem Proving

in the future

- plus Model Checking & Theorem Proving integrated

Theorem Proving for Process Algebra

CSP

HOL-CSP (Tej/Wolf 1997)

Schneider/Dutertre (2001)

CSP-Prover (2005)

CCS

Nesi (1992)

μCRL/ACP

van de Pol (2001)

Badban et al (2005)

π-calculus

Röckl/Hirschkoff (2003)

Bengtson/Parrow (2007)

Theorem Proving yields new qualities

Theorem Proving yields new qualities

Computer Science

- Verify language design, e.g., of CSP
- Verify proof principles, e.g., of CSP

Theorem Proving yields new qualities

Computer Science

- Verify language design, e.g., of CSP
- Verify proof principles, e.g., of CSP

Applications

- Increase the level of trust into
  - the tool itself
    (model checkers are known to produce incorrect results)
  - the overall proof, e.g., by proving abstractions with a tool
- ‘New’ properties accessible: richer input language

Cost of Theorem Proving

negative aspect

interactive proofs require lots of time and a certain expertise

Cost of Theorem Proving

negative aspect

interactive proofs require lots of time and a certain expertise

positive perspective

increasingly powerful tactics automatise simple proof steps

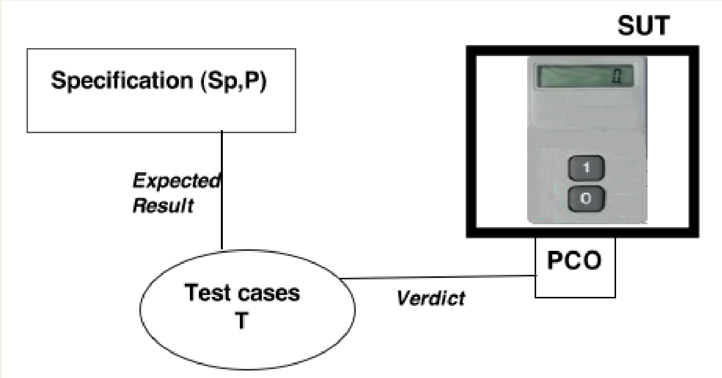
CSP-Prover in Practice

Turn Semantics into Syntactic Proof Principles

- semantical equation $\leadsto$ algebraic law
  - how to prove them?
  - how to apply them?
- solution of equations $\leadsto$ fixed point induction
- definition of deadlock $\leadsto$ proof by abstraction

How to do equational reasoning?

Example: One bit calculator



$$(((Calculator \parallel Test)\ [[MyRenaming]]) \mathbin{[\![}\{a\}\mathbin{]\!]} Count3) \setminus \{a\}$$

$$=_{\mathcal{T}}$$

$$OK \rightarrow Stop$$

Only the 1st proof step:

$$Calculator \parallel Test =_{\mathcal{T}} Test$$

where

$$Calculator = ?x : Button \to ?y : Button \to Display!(x + y) \to Skip$$

$$Test = Button!0 \to Button!1 \to Display!1 \to Stop$$

Three helpful lemmas

1. $$(?x : \{Button!0, Button!1\} \to P(x)) \parallel (Button!y \to Q)$$
$$=_{\mathcal{F}} \; Button!y \to (P(Button!y) \parallel Q)$$

2. $$(Display!x \to P) \parallel (Display!x \to Q)$$
$$=_{\mathcal{F}} \; Display!x \to (P \parallel Q)$$

3. $SKIP \parallel STOP =_{\mathcal{F}} STOP$

Remarks on Equational Reasoning

Strategy

- provide intermediate lemmas
- simplification and reduction to head normal form will do

Remarks on Equational Reasoning

Strategy

- provide intermediate lemmas
- simplification and reduction to head normal form will do

Dream: do all proofs by (conditional) equational reasoning.

Obstacle: Of the 4 main CSP models, only for one a complete axiomatic semantics is known.

Remarks on Equational Reasoning

Strategy

- provide intermediate lemmas
- simplification and reduction to head normal form will do

Dream: do all proofs by (conditional) equational reasoning.

Obstacle: Of the 4 main CSP models, only for one a complete axiomatic semantics is known.

Refinement $\sqsubseteq$ can be encoded in terms of $=$

$\leadsto$ same treatment as $=$

Proof Principles have their preconditions

A specification:

$$X = (a \to X) \,\Box\, X$$

Proof Principles have their preconditions

A specification:

$$X = (a \to X) \,\Box\, X$$

Two canditates as solution over $\mathcal{T}$:

$$\begin{aligned} Q_a &= (a \to Q_a) \\ Q_{ab} &= (a \to Q_{ab}) \,\Box\, (b \to Q_{ab}) \end{aligned}$$

Proof Principles have their preconditions

A specification:

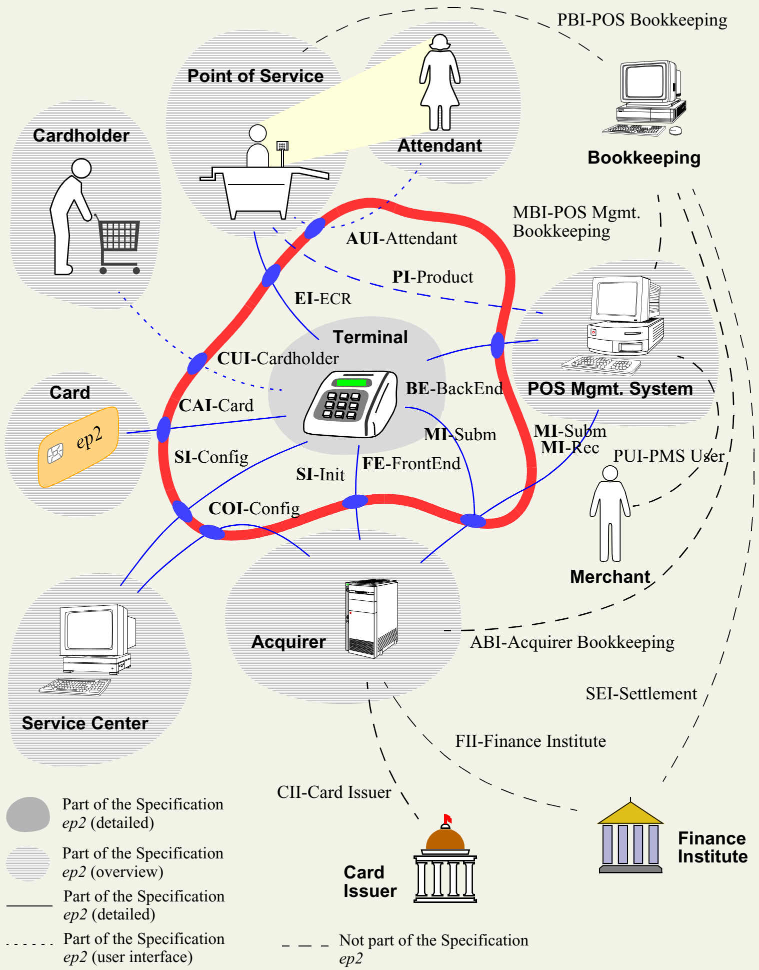
$$X = (a \to X) \,\Box\, X$$

Two canditates as solution over $\mathcal{T}$:

$$\begin{aligned} Q_a &= (a \to Q_a) \\ Q_{ab} &= (a \to Q_{ab}) \,\Box\, (b \to Q_{ab}) \end{aligned}$$

Two potential proof principles: metric-fpi and cpo-fpi

Deadlock freedom



From semantical properties to syntactical characterization

Semantic definition

P is *deadlock-free* iff $\forall\, s \in \Sigma^* \bullet (s, \Sigma^{\checkmark}) \notin \mathit{failures}(P)$

From semantical properties to syntactical characterization

Semantic definition

P is *deadlock-free* iff $\forall\, s \in \Sigma^* \bullet (s, \Sigma^{\checkmark}) \notin \mathit{failures}(P)$

Syntactic characterization $DF^{\checkmark} \sqsubseteq_{\mathcal{F}} P$

From semantical properties to syntactical characterization

Semantic definition

P is *deadlock-free* iff $\forall\, s \in \Sigma^* \bullet (s, \Sigma^{\checkmark}) \notin \mathit{failures}(P)$

Syntactic characterization $DF^{\checkmark} \sqsubseteq_{\mathcal{F}} P$

Practical use

$$DF^{\checkmark} \sqsubseteq_{\mathcal{F}} \mathit{Abstraction} \sqsubseteq_{\mathcal{F}} EP2 - \mathit{Dialog}$$

From semantical properties to syntactical characterization

Semantic definition

P is *deadlock-free* iff $\forall\, s \in \Sigma^* \bullet (s, \Sigma^{\checkmark}) \notin \mathit{failures}(P)$

Syntactic characterization $DF^{\checkmark} \sqsubseteq_{\mathcal{F}} P$

Practical use

$$DF^{\checkmark} \sqsubseteq_{\mathcal{F}} Abstraction \sqsubseteq_{\mathcal{F}} EP2 - Dialog$$

Link Correctnes proof of the characterization in CSP-Prover

A fundamental weakness of this approach

- (currently?) ‘small’ & ‘concrete’ problems only
- ‘Abstraction’ expands parallelism into non-determinism
  $\leadsto$ exponentiell growth in the number of parallel operators

A fundamental weakness of this approach

- (currently?) 'small' & 'concrete' problems only
- 'Abstraction' expands parallelism into non-determinism
  $\leadsto$ exponentiell growth in the number of parallel operators

DFP-package

- semantical – hard to use
- scales up to arbitrarily large problems
- can deal with parametrized problems

Summary and Future Work

Summary

- CSP proof principles & their support in CSP-Prover
  - equational reasoning
  - fixed-point induction
  - proof by abstraction
- Theorem Proving for CSP requires knowledge of its proof principles
- CSP-Prover 4.0 works well

Future Work

- Extend CSP-Prover
  - tactics and proof principles
  - models: $\mathcal{N}, R$
- Integrate FDR and CSP-Prover (Bremen University)
- Case studies
  - Analysing Parallel & Distributed Algorithms (University of Pretoria)
  - Electronic Warehouse (Qinetic, UK)
  - Logistics?

Formalisation in CSP

```
n: Nat
startConfig: array [0..(n-1)] of Nat  %% all entries even

channel
  c.Nat: Nat
```

```
let
  Pass (p:Nat,candies:Nat)    =
      (    c.(p     mod n) ! candies div 2
        -> c.(p+n-1 mod n) ? x: Nat
        -> Fill (p, (candies div 2) + x ) )
    []
      (    c.(p+n-1 mod n) ? x: Nat
        -> c.(p mod n]       ! candies div 2
        -> Fill (p, (candies div 2) + x ) )
  Fill (p:Nat,candies:Nat) = if even(c) then Pass(i,candies)
                                         else Pass(i,candies+1)
in
  ||_{p=0}^(n-1) Pass(p,startConfig[i])
```