A Complete Axiomatic Semantics for the CSP Stable-Failures Model

Yoshinao Isobe, AIST, Japan

in cooperation with
Markus Roggenbach, University of Wales Swansea, UK

CSP Prover

TPP 2006 (30/11/2006) (also see CONCUR 2006)

Overview

1 Introduction

- Process algebra
- Motivation

2 Non-determinism (ND)

- Syntax of unbounded ND

3 Axiom system $\mathcal{A}_{\mathcal{F}}$

- Differences from finite version
- Sequentialisation and Normalisation

4 CSP-Prover

- A deep-encoding of CSP in Isabelle

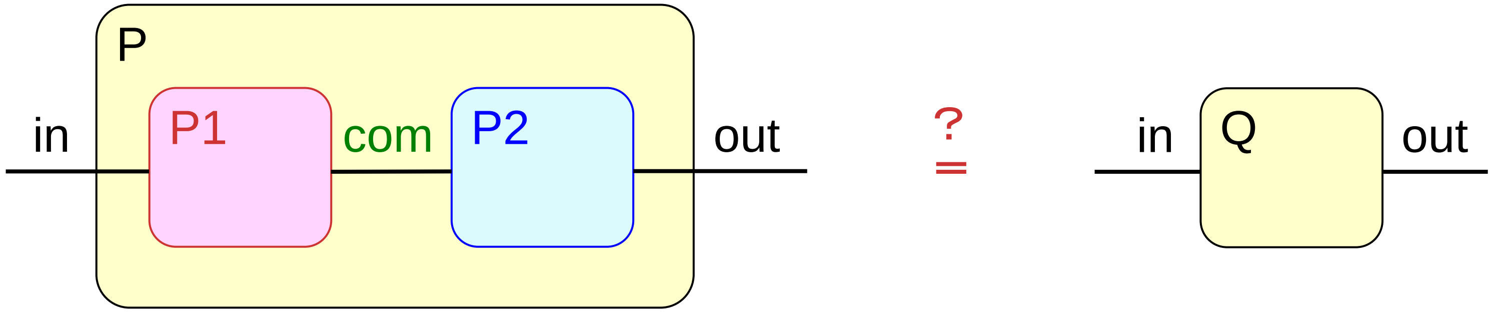
5 Conclusion

- Summary and future work

Introduction

Process algebra

a formal framework to describe and analyze concurrent processes.



P = (P1 |[com]| P2)＼com

P1 = in → com → STOP

P2 = com → out → STOP

P $\stackrel{?}{=}$ Q

Q = in → out → STOP

---

3 styles of semantics

- Operational semantics
- Denotational semantics
- Axiomatic semantics

Operational semantics

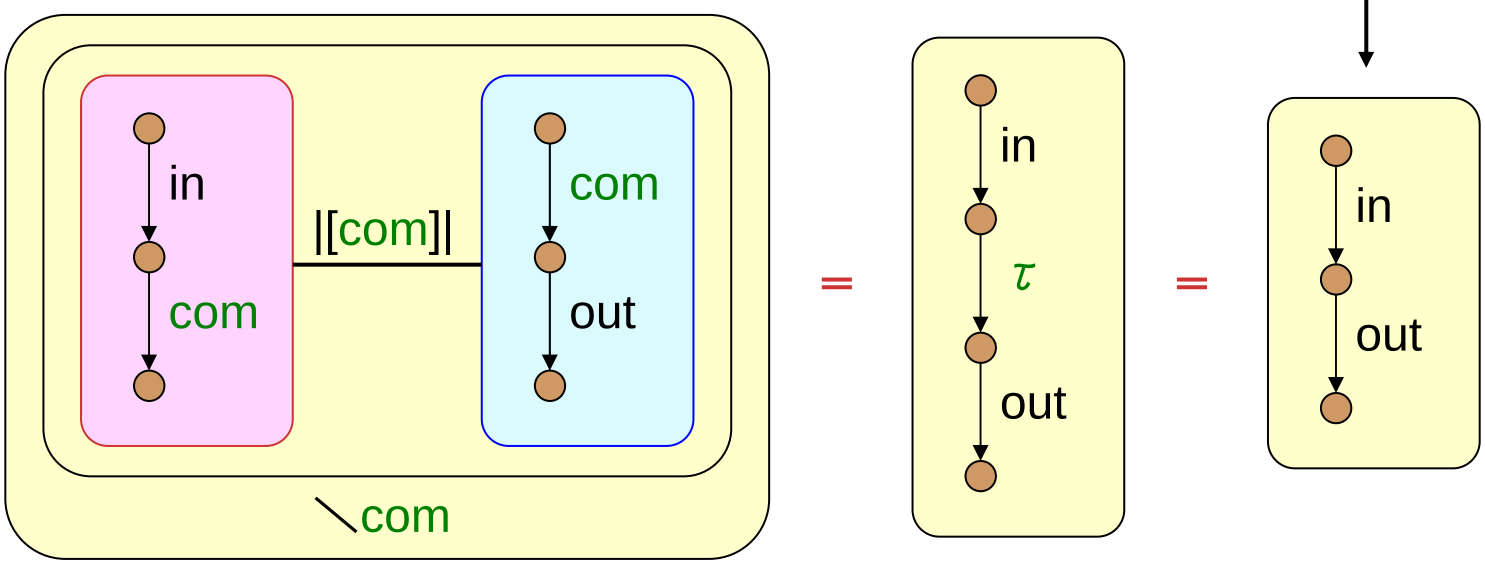
P = (P1 |[com]| P2)＼com

P1 = in → com → STOP

P2 = com → out → STOP

Q = in → out → STOP

Graph



Denotational semantics

P = (P1 |[com]| P2)＼com

P1 = in → com → STOP

P2 = com → out → STOP

Q = in → out → STOP

Domain
(traces model)

traces(Q) = { 〈〉, 〈in〉, 〈in.out〉}

||

traces(P) = { (t_1 |[com]| t_2)＼com | $t_1 \in$ traces(P1), $t_2 \in$ traces(P2)}
= { 〈〉, 〈in〉, 〈in.out〉}

traces(P1) = { 〈〉, 〈in〉, 〈in.com〉}

traces(P2) = { 〈〉, 〈com〉, 〈com.out〉}

Denotational semantics

P = (P1 |[com]| P2)＼com

P1 = in → com → STOP

P2 = com → out → STOP

Q = in → out → STOP

Domain (stable-failures model)

traces(Q) = { ⟨⟩, ⟨in⟩, ⟨in.out⟩}

failures(Q) = {(⟨⟩,{out}), (⟨in⟩,{in}), (⟨in.out⟩,{in,out})}

||

traces(P) = { ⟨⟩, ⟨in⟩, ⟨in.out⟩}

failures(P) = { (⟨⟩, {out}), (⟨in⟩, {in}), (⟨in.out⟩, {in,out})}

refusals (refused events)

Axiomatic semantics

P = (P1 |[com]| P2)＼com

P1 = in → com → STOP

P2 = com → out → STOP

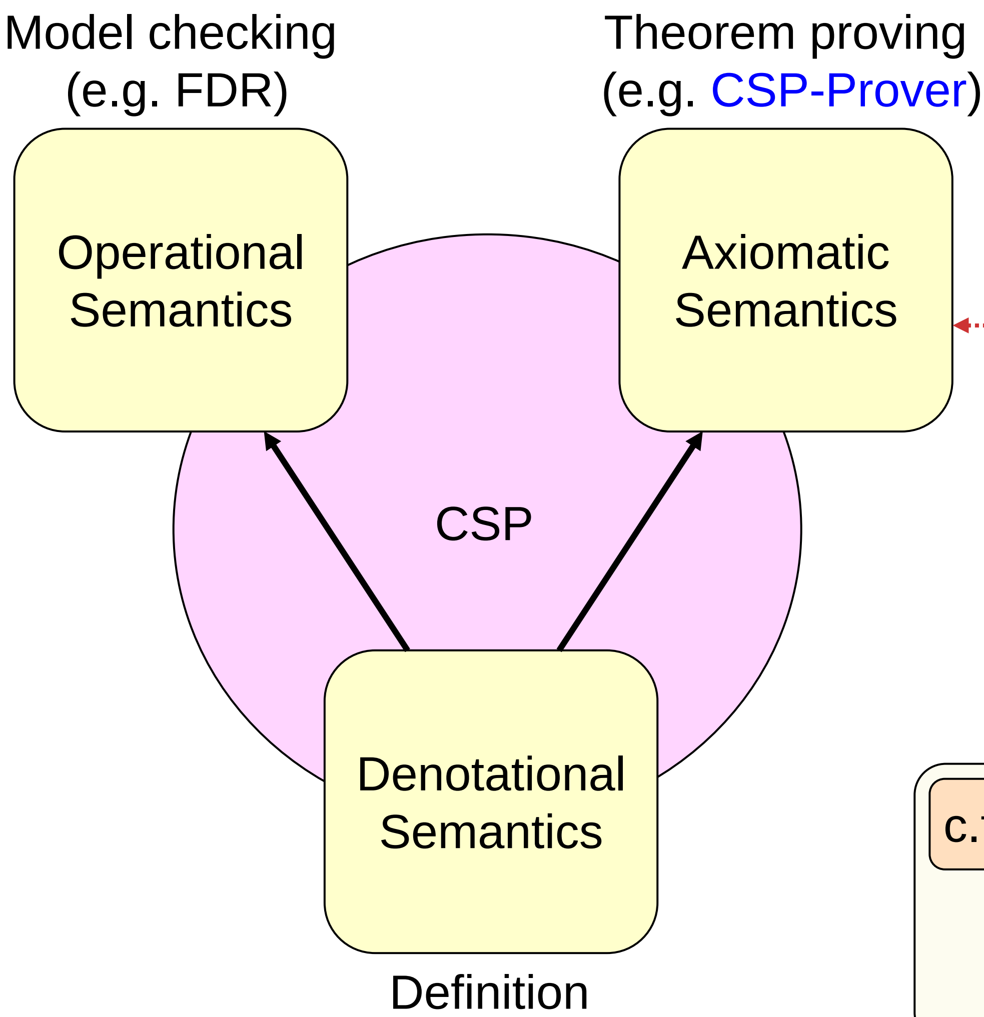
P = (P1 |[com]| P2)＼com
= ((in → com → STOP) |[com]| (com → out → STOP))＼com
= (in → ((com → STOP) |[com]| (com → out → STOP)))＼com — by $[para_2]$
= in → ((com → STOP) |[com]| (com → out → STOP))＼com — by $[hide_2]$
= in → (com → (STOP |[com]| (out → STOP)))＼com — by $[para_1]$
= in → (STOP |[com]| (out → STOP))＼com — by $[hide_1]$
= in → (out → (STOP |[com]| STOP))＼com
⋮
= in → out → (STOP |[com]| STOP)＼com
= in → out → STOP＼com
= in → out → STOP = Q

Q = in → out → STOP

axiom system:

$[para_1]$	(a → P) |[a]| (a → Q) = a → (P |[a]| Q)
$[para_2]$	(a → P) |[b]| (b → Q) = a → (P |[b]| (b → Q))
$[hide_1]$	(a → P)＼a = P＼a
$[hide_2]$	(b → P)＼a = b → (P＼a)
⋮	

Process algebra (CSP)



open question of CSP

Completeness ?
for unbounded nondeterministic CSP over an arbitrary alphabet

c.f. The best known results apply for finitely nondeterministic CSP over a finite alphabet.
[Brooks(1983) , Roscoe (1998)]

Process algebra (CSP)

M

Our question is:

Is it possible to prove the equality of two CSP-processes by algebraic laws without using denotational semantics?

stic
et

Semantics

Definition

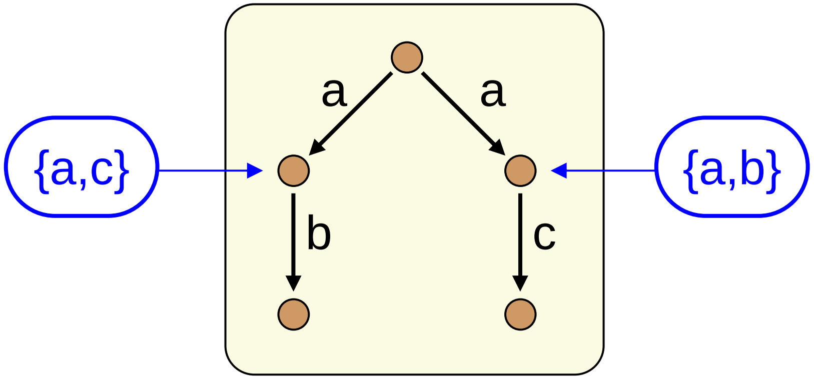
c.f. The best known results apply for finitely nondeterministic CSP over a finite alphabet.
[Brooks(1983) , Roscoe (1998)]

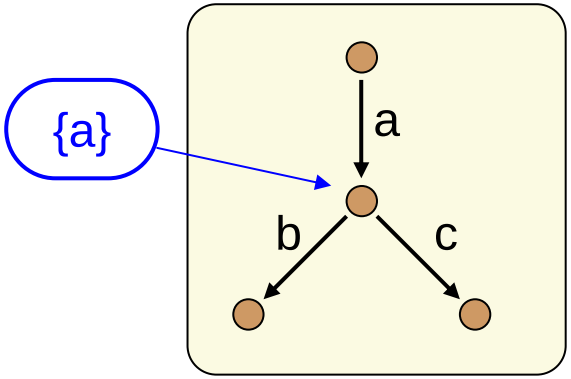
Non-determinism

External choices

external choice □

$a \to b \to STOP \; \square \; a \to c \to STOP \quad \neq_{\mathcal{F}} \quad a \to (b \to STOP \; \square \; c \to STOP)$





We focus on the stable-failures model suitable for describing infinite systems and deadlock analysis.

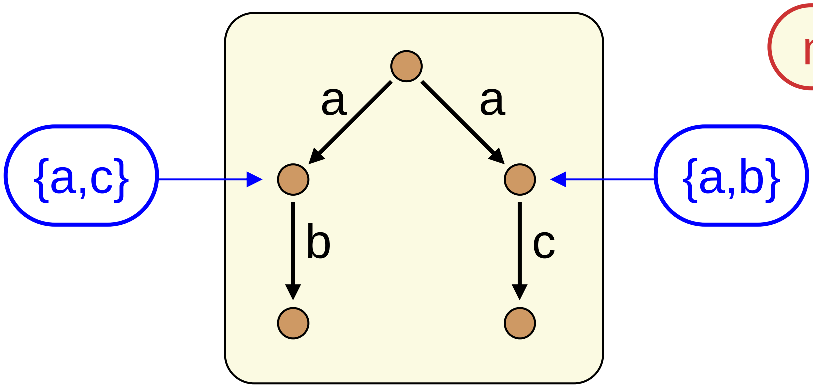
Internal choices

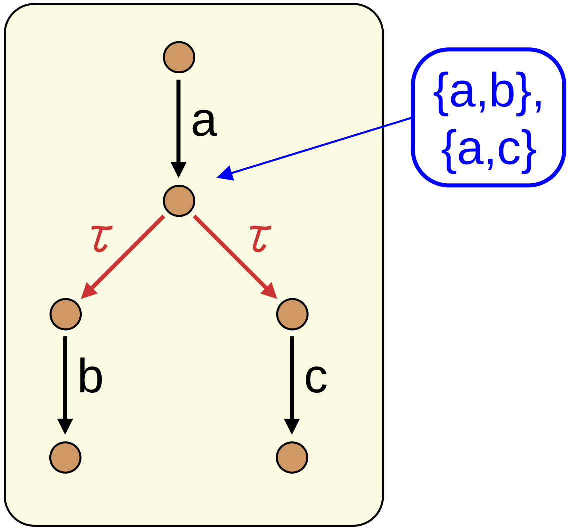
internal choice ⊓

$$a \rightarrow b \rightarrow STOP \;\square\; a \rightarrow c \rightarrow STOP \quad =_{\mathcal{F}} \quad a \rightarrow (b \rightarrow STOP \;\sqcap\; c \rightarrow STOP)$$

note

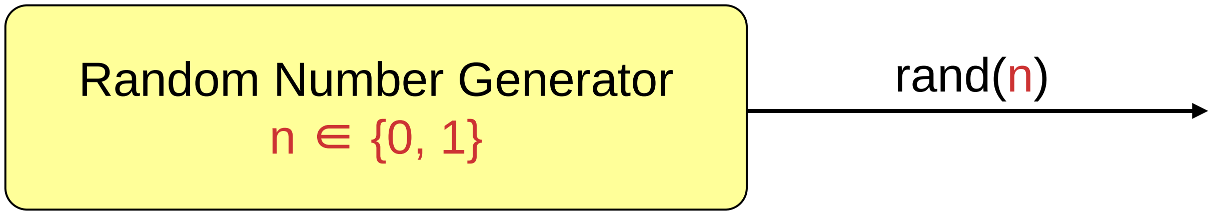
note





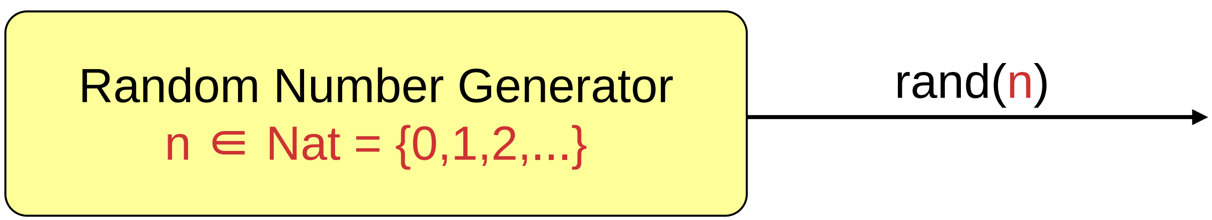
Unbounded non-determinism

binary internal choice



$$RNG = (rand(0) \rightarrow STOP) \sqcap (rand(1) \rightarrow STOP)$$

general internal choice



$$RNG = \sqcap \{rand(n) \rightarrow STOP \mid n \in Nat\}$$

a set of processes

Standard CSP

Syntax

a set of processes

Proc ::= STOP | a → Proc | Proc □ Proc | ⊓ (Proc Set) | …

Isabelle type

'a : type of alphabet (events) Σ

datatype 'a proc = STOP
| Act_prefix "'a" "'a proc" (_ → _)
| Ext_choice "'a proc" "'a proc" (_ □ _)
| G_Int_choice "'a proc set" (⊓ _)
| …

⇒ cardinality mismatch ✕

CSP_{TP}

Syntax

process function

Proc ::= STOP | a → Proc | Proc □ Proc | ⊓ (Proc Fun) | …

Isabelle type

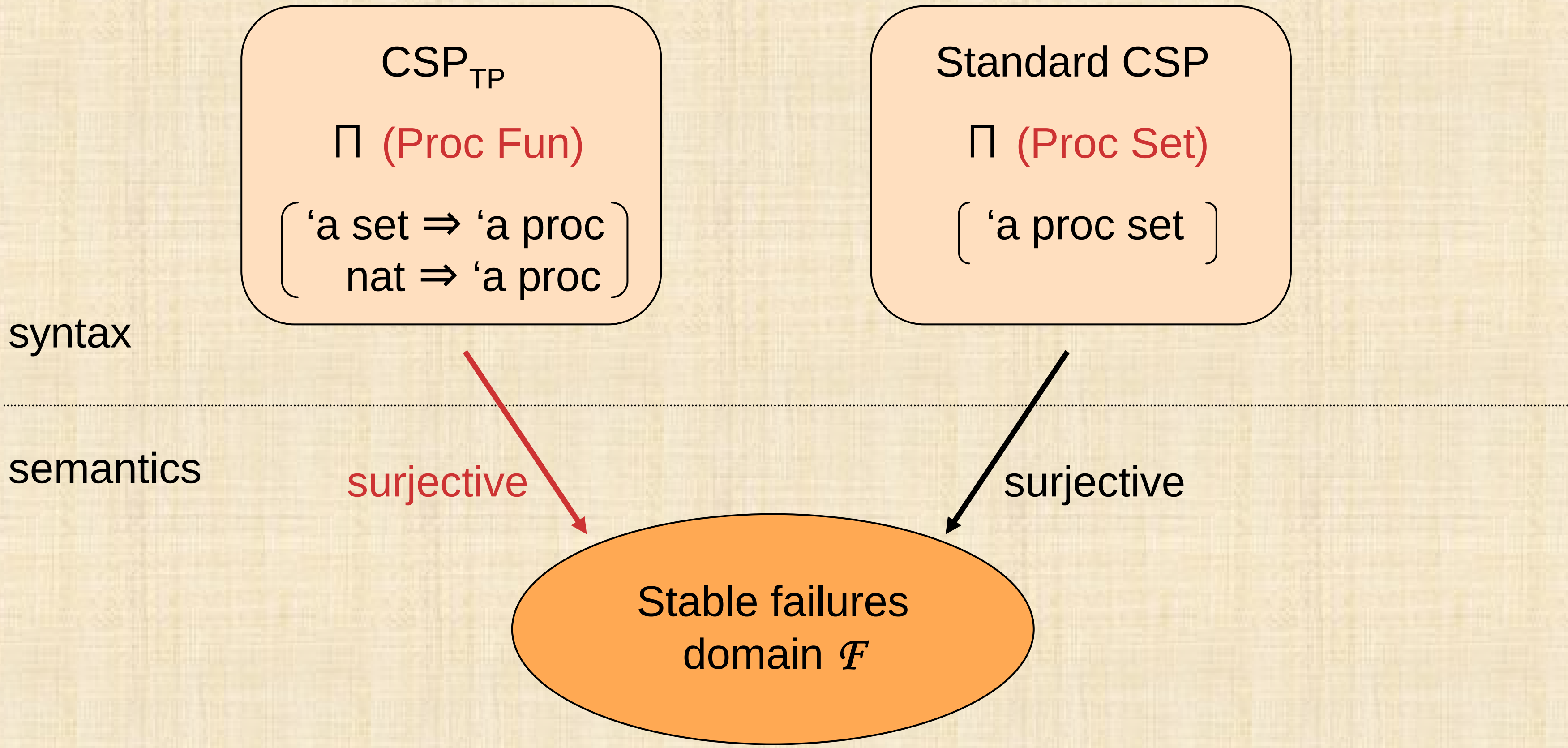
datatype ‘a proc = STOP

| Act_prefix	“ ’a” “ ‘a proc”	(_ → _)
| Ext_choice	“ ’a proc” “ ‘a proc”	(_ □ _)
| Set_Int_choice	“ ‘a set ⇒ ‘a proc”	($\sqcap_{set}$ _)
| Nat_Int_choice	“ nat ⇒ ‘a proc”	($\sqcap_{nat}$ _)
| …		

note these types

Relation to 'Standard CSP'

Expressive power



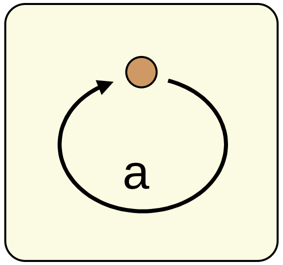
Div: The bottom element (in semantic domain)

Recursive processes

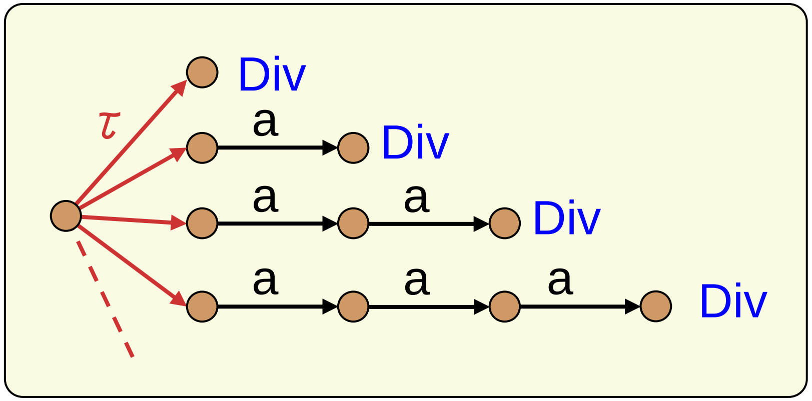
$$\text{Loop} = a \rightarrow \text{Loop}$$

$$\text{Loop}^{(0)} = \text{Div}$$
$$\text{Loop}^{(n+1)} = a \rightarrow \text{Loop}^{(n)}$$

$$\text{Loop} =_{\mathcal{F}} \sqcap \{\text{Loop}^{(n)} \mid n \in \text{Nat}\}$$



$=_{\mathcal{F}}$



Axiom system

Axiom system $\mathcal{A}_{\mathcal{F}}$

axiom system $\mathcal{A}_{\mathcal{F}}$

$\mathcal{A}_{\mathcal{F}}$ is sound and complete for the stable failures equivalence over unbounded nondeterministic processes with an arbitrary alphabet.

$$\forall P,Q \in Proc. \quad \mathcal{A}_{\mathcal{F}} \vdash P = Q \iff P =_{\mathcal{F}} Q$$

Important differences from the standard axioms for finite processes appear in the laws for

(1) parallel composition in combination with timeout (corrected)

(2) internal choice in combination with Skip (extended with infinity)

(3) depth restriction operator (new)

Depth restriction

$P \downarrow n$: depth restriction by the n^{th} step

examples

$(Stop) \downarrow 2 =_{\mathcal{F}} STOP$

$(a_1 \to Stop) \downarrow 2 =_{\mathcal{F}} a_1 \to STOP$

$(a_1 \to a_2 \to Stop) \downarrow 2 =_{\mathcal{F}} a_1 \to a_2 \to Div$

$(a_1 \to a_2 \to a_3 \to Stop) \downarrow 2 =_{\mathcal{F}} a_1 \to a_2 \to Div$

$(a_1 \to a_2 \to a_3 \to a_4 \to Stop) \downarrow 2 =_{\mathcal{F}} a_1 \to a_2 \to Div$

all the executions are cut off at the 2^{nd} step

$P =_{\mathcal{F}} \sqcap_{nat} (\lambda n \bullet (P \downarrow n))$

How to normalise

any process

key point :
remove Hiding operators by a function recursively defined on the process structure.

full sequential form

key point :
normalise $((\sqcap_{set} P(X)) \downarrow n)$ by a function recursively defined on the depth n.

(extended) full normal form

note

Induction on the process structure cannot be applied to a family of processes $P(X)$ $(X \subseteq \Sigma)$

How to normalise

any pr

$\forall X \subseteq \Sigma.\ P(X) \in FNF$

$\sqcap_{set} P(X)$ can be normalized if Σ is finite.

However, Σ can be infinite!

full sequential form

key point :
normalise $((\sqcap_{set} P(X)) \downarrow n)$ by a function recursively defined on the depth n.

(extended) full normal form

note

Induction on the process structure cannot be applied to a family of processes $P(X)$ $(X \subseteq \Sigma)$

How to normalise

note

$P \downarrow n =_{\mathcal{F}} Q \downarrow n \not\Rightarrow (P \setminus X) \downarrow n =_{\mathcal{F}} (Q \setminus X) \downarrow n$

any process

key point :
remove Hiding operators by a function recursively defined on the process structure.

full sequential form

key point :
normalise $((\sqcap_{set} P(X)) \downarrow n)$ by a function recursively defined on the depth n.

(extended) full normal form

note

Induction on the process structure cannot be applied to a family of processes $P(X)$ $(X \subseteq \Sigma)$

Full Sequential form

Full Sequential Form (FSF)

FSF contains only "sequential" operators such as $\Box$, !!, and Stop.

The following function Seq: Proc→FSF can be recursively defined over the process structure.

Theorem 3

$$\forall P \in \mathrm{Proc}.\ \mathcal{A}_{\mathcal{F}} \vdash P = \mathrm{Seq}(P)$$

This theorem can be proven by structural induction on P.

note The sequential process Seq(P) cannot be necessarily automatically computed because Seq(P) often needs infinite computations, for example

$$\mathrm{Seq}(\sqcap s \bullet P(s))$$

requires to compute Seq(P(s)) for all $s \in S$, where S may be infinite.

Full Normal form

Syntactic equality?

$\sqcap s \bullet (\sqcap s' \bullet P_{seq}(s,s')) \in FSF$

$\sqcap s' \bullet (\sqcap s \bullet P_{seq}(s,s')) \in FSF$

(semantically equal but syntactically different)

Full Normal Form (FNF) (similar to the standard FNF)

FNF is a more specialized form than FSF, for giving the syntactic equality.

Theorem 4

$\forall P,Q \in FNF.\ \ P =_{\mathcal{F}} Q \iff P \equiv Q$ (syntactic equality)

Full Normal form

The following function Norm: FSF→FNF can be recursively defined on the depth n and the structure over FSF.

Lemma 2

$\forall P \in FSF.\ \mathcal{A}_{\mathcal{F}} \vdash P \downarrow n = Norm_{(n)}(P)$

This theorem can be proven by the induction on n and structural induction on P.

P may be (!! s:S • P'(s))

FNF does not capture all processes

There is no function Norm' such that $\forall P \in FSF.\ \mathcal{A}_{\mathcal{F}} \vdash P = Norm'(P)$

Theorem 5

$\exists P \in FSF.\ \forall P' \in FNF.\ P \neq_{\mathcal{F}} P'$

Extended Full Normal Form

reminder

$P =_{\mathcal{F}} \sqcap n \bullet (P \downarrow n)$

Extended Full Normal Form (XFNF)

$P = \sqcap n \bullet P'(n)$ if
(1) $\forall n.\ P'(n) \in FNF$ and
(2) $\forall n.\ P \downarrow n =_{\mathcal{F}} P'(n)$

infinite internal choice over fully normalised processes for finite depths

Theorem 6

$\forall P,Q \in XFNF.\ \ P =_{\mathcal{F}} Q \Leftrightarrow P \equiv Q$ (syntactic equality)

Theorem 7

$\forall P \in Proc.\ \ \exists P' \in XFNF.\ \mathcal{A}_{\mathcal{F}} \vdash P = Xnorm(P)$

$Xnorm(P) \equiv \sqcap n \bullet (Norm_{(n)}(Seq(P)))$

Completeness

Corollary $\forall P,Q \in Proc.\ P =_{\mathcal{F}} Q \Rightarrow \mathcal{A}_{\mathcal{F}} \vdash P = Q$

Let $P =_{\mathcal{F}} Q$, then

$$\mathcal{A}_{\mathcal{F}} \vdash P = Xnorm(P) \equiv Xnorm(Q) = Q$$

Theorem 6 $\forall P,Q \in XFNF.\ \ P =_{\mathcal{F}} Q \ \Leftrightarrow\ P \equiv Q$ (syntactic equality)

Theorem 7 $\forall P \in Proc.\ \ \exists P' \in XFNF.\ \mathcal{A}_{\mathcal{F}} \vdash P = Xnorm(P)$

$$Xnorm(P) \equiv \sqcap n \bullet (Norm_{(n)}(Seq(P)))$$

CSP-Prover

CSP-Prover

CSP-Prover: a deep encoding of CSP in the generic theorem prover Isabelle

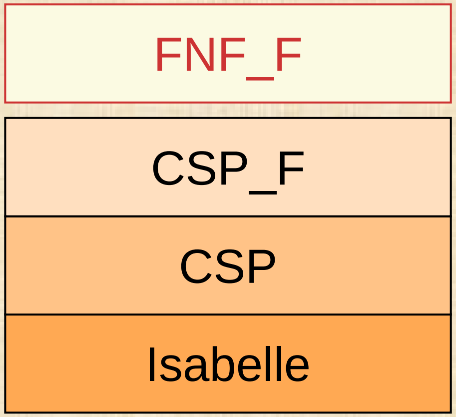
includes fixed point theorems, definitions of syntax and semantics, CSP-laws, semi-automatic proof tactics, etc.

- Verification of infinite state systems

  e.g. EP2 (an electronic payment system)

- Establishing new theorems on CSP

  e.g. Soundness and completeness of $\mathcal{A}_{\mathcal{F}}$



References:
1. Y.Isobe and M.Roggenbach, A Generic Theorem Prover of CSP refinment, TACAS 2005, LNCS 3440, pp.108-123, 2005
2. Y.Isobe and M.Roggenbach, A complete axiomatic semantics for CSP stable failures model, CONCUR 2006, LNCS 4237, pp.158-172, 2006

Web-site: http://staff.aist.go.jp/y-isobe/CSP-Prover/CSP-Prover.html

Conclusion

Summary and Future Work

Summary

1. Complete axiomatic semantics of the stable failures model
2. Our CSP dialect is expressive with respect to the stable failures model
3. Implementation & Verification of all results in CSP-Prover
4. Correction of two well-known step laws

   The errors as well as our corrections have been approved by Bill Roscoe, Oxford.

Future work

1. Improve proof tactics in CSP-Prover based on the normal forms
2. Develop completeness results for other CSP models