CSP-CASL: Semantics, Application, Tools

Markus Roggenbach (Swansea)

June 2006

Overview

CSP-CASL: Semantics

Modelling ep2 in CSP-CASL

Towards tool support for CSP-CASL

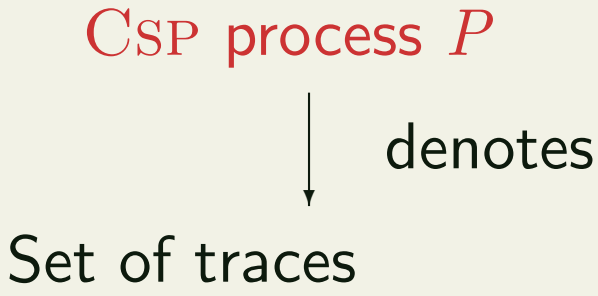
CSP-CASL: Semantics

CSP and CASL

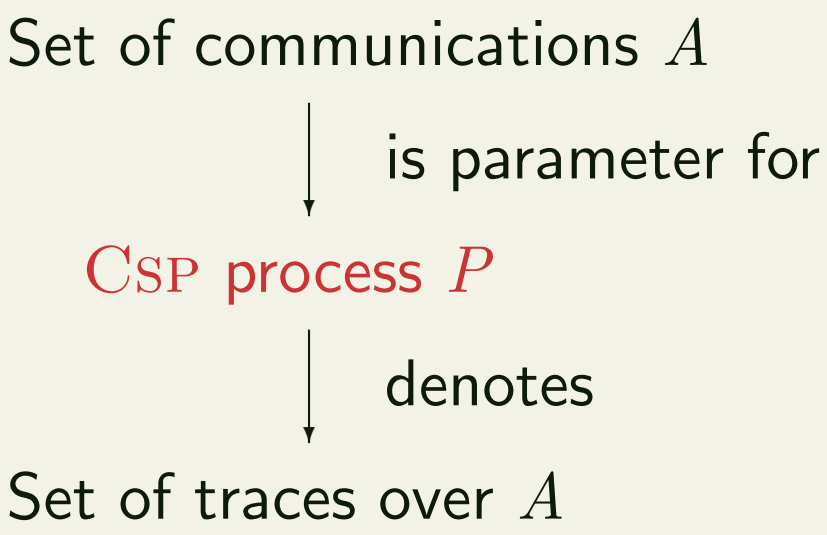
CSP and CASL

CSP process P

CSP and CASL



CSP and CASL


Set of communications A
is parameter for
CSP process P
denotes
Set of traces over A

CSP and CASL

Set of communications A

↓ is parameter for

CSP process P

↓ denotes { Tests over A : $a = b?, a \in X?, aRb?$

Set of traces over A

CSP and CASL

CASL specification Sp

Set of communications A

↓ is parameter for

CSP process P

↓ denotes { Tests over A : $a = b?, a \in X?, aRb?$

Set of traces over A

CSP and CASL

CASL specification Sp

↓ has as semantics

Subsorted signature $(S, TF, PF, P, \leq)$
Class of Σ-models M; $(M_s)_{s \in S}$ family of carrier sets

Set of communications A

↓ is parameter for

CSP process P

↓ denotes { Tests over A : $a = b?, a \in X?, aRb?$

Set of traces over A

CSP and CASL

CASL specification Sp

↓ has as semantics

Subsorted signature $(S, TF, PF, P, \leq)$
Class of Σ-models M; $(M_s)_{s \in S}$ family of carrier sets

⇳ Interface Problem ⇳

Set of communications A

↓ is parameter for

CSP process P

↓ denotes { Tests over A : $a = b?, a \in X?, aRb?$

Set of traces over A

From models to alphabets

CASL
SubPFOL$^{=}$

CSP
alphabet A

From models to alphabets

CASL
SubPFOL$^{=}$

CSP
alphabet A

$(M_s)_{s \in S}$

$(\biguplus_{s \in S} M_s \cup \{\bot_s\})_{/\sim}$

From models to alphabets

CASL
SubPFOL$^{=}$

CSP
alphabet A

Tests
$a = b?,\ \ a \in X?,\ \ aRb?$

$(M_s)_{s \in S}$

$(\biguplus_{s \in S} M_s \cup \{\bot_s\})_{/\sim}$

From models to alphabets

CASL
SubPFOL$^{=}$

CSP
alphabet A

subsorted signatures
$\Sigma = (S, TF, PF, P, \leq)$

$FO(\Sigma)$

Tests
$a = b?,\ \ a \in X?,\ \ aRb?$

$(M_s)_{s \in S}$

$(\biguplus_{s \in S} M_s \cup \{\bot_s\})_{/\sim}$

From models to alphabets

CASL	'hidden bridge'	CSP
SubPFOL$^=$	FinCommSubPFOL$^=$	alphabet A

subsorted signatures
$\Sigma = (S, TF, PF, P, \leq)$

$FO(\Sigma)$

Tests
$a = b?,\ \ a \in X?,\ \ aRb?$

$(M_s)_{s \in S}$

$(\biguplus_{s \in S} M_s \cup \{\bot_s\})_{/\sim}$

From models to alphabets

CASL SubPFOL$^=$	'hidden bridge' FinCommSubPFOL$^=$	CSP alphabet A
subsorted signatures $\Sigma = (S, TF, PF, P, \leq)$	subsorted signatures S finite, local top-elements	
$FO(\Sigma)$		Tests $a = b?,\ a \in X?,\ aRb?$
$(M_s)_{s \in S}$		$(\biguplus_{s \in S} M_s \cup \{\bot_s\})_{/\sim}$

From models to alphabets

CASL SubPFOL$^=$	'hidden bridge' FinCommSubPFOL$^=$	CSP alphabet A
subsorted signatures $\Sigma = (S, TF, PF, P, \leq)$	subsorted signatures S finite, local top-elements	
$FO(\Sigma)$	$FO'(\Sigma)$ e.g., $t_1 = t_2$ for t_i of arbitrary sort	Tests $a = b?,\ \ a \in X?,\ \ aRb?$
$(M_s)_{s \in S}$		$(\biguplus_{s \in S} M_s \cup \{\bot_s\})_{/\sim}$

From models to alphabets

CASL SubPFOL$^=$	'hidden bridge' FinCommSubPFOL$^=$	CSP alphabet A
subsorted signatures $\Sigma = (S, TF, PF, P, \leq)$	subsorted signatures S finite, local top-elements	
$FO(\Sigma)$	$FO'(\Sigma)$ e.g., $t_1 = t_2$ for t_i of arbitrary sort	Tests $a = b?,\ a \in X?,\ aRb?$
$(M_s)_{s \in S}$	$(M_s \cup \{\bot_s\})_{s \in S}$	$(\biguplus_{s \in S} M_s \cup \{\bot_s\})_{/\sim}$

From models to alphabets

CASL SubPFOL$^=$		'hidden bridge' FinCommSubPFOL$^=$	CSP alphabet A
subsorted signatures $\Sigma = (S, TF, PF, P, \leq)$	$\xleftarrow{\Phi}$	subsorted signatures S finite, local top-elements	
$FO(\Sigma)$	$\xleftarrow{\alpha}$	$FO'(\Sigma)$ e.g., $t_1 = t_2$ for t_i of arbitrary sort	Tests $a = b?,\ a \in X?,\ aRb?$
$(M_s)_{s \in S}$	$\xrightarrow{\beta}$	$(M_s \cup \{\bot_s\})_{s \in S}$	$(\biguplus_{s \in S} M_s \cup \{\bot_s\})_{/\sim}$

Simple Representation

From models to alphabets

CASL SubPFOL$^=$	‘hidden bridge’ FinCommSubPFOL$^=$	CSP alphabet A

subsorted signatures $\xleftarrow{\Phi}$ subsorted signatures
$\Sigma = (S, TF, PF, P, \leq)$ S finite, local top-elements

$FO(\Sigma)$ $\xleftarrow{\alpha}$ $FO'(\Sigma)$ $\xleftarrow{\texttt{for}}$ Tests

e.g., $t_1 = t_2$ for t_i of arbitrary sort

$a = b?,\ a \in X?,\ aRb?$

$(M_s)_{s \in S}$ $\xrightarrow{\beta}$ $(M_s \cup \{\bot_s\})_{s \in S}$ $\xrightarrow{\texttt{emb}}$ $(\biguplus_{s \in S} M_s \cup \{\bot_s\})_{/\sim}$

Simple Representation **Characterization**

A simple example

- CSP-CASL specification:

$$\begin{array}{lll} \textbf{data} & \textbf{sorts} & S, T \\ & \textbf{ops} & c : S;\ d : T \\ \textbf{process} & c \rightarrow SKIP & \|\ \ d \rightarrow SKIP \end{array}$$

A simple example

- CSP-CASL specification:

$$\begin{array}{lll} \textbf{data} & \textbf{sorts} & S, T \\ & \textbf{ops} & c : S;\ d : T \\ \textbf{process} & & c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP \end{array}$$

- CSP-CASL semantics (for one model M of the data part):

$$traces([\![c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP]\!]_{\emptyset})$$

A simple example

- CSP-CASL specification:

$$\begin{array}{lll} \textbf{data} & \textbf{sorts} & S, T \\ & \textbf{ops} & c : S;\ d : T \\ \textbf{process} & & c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP \end{array}$$

- CSP-CASL semantics (for one model M of the data part):

$$\begin{array}{rl} & traces([\![c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP]\!]_{\emptyset}) \\ = & traces(\mathtt{emb}_S(\emptyset^{\sharp}_S(c)) \rightarrow SKIP \ \ || \ \ \mathtt{emb}_T(\emptyset^{\sharp}_T(d)) \rightarrow SKIP) \end{array}$$

A simple example

- CSP-CASL specification:

$$\begin{array}{lll} \textbf{data} & \textbf{sorts} & S, T \\ & \textbf{ops} & c : S;\ d : T \\ \textbf{process} & & c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP \end{array}$$

- CSP-CASL semantics (for one model M of the data part):

$$\begin{array}{rl} & traces([\![c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP]\!]_\emptyset) \\ = & traces(\mathtt{emb}_S(\emptyset^\sharp_S(c)) \rightarrow SKIP \ \ || \ \ \mathtt{emb}_T(\emptyset^\sharp_T(d)) \rightarrow SKIP) \\ = & traces(\mathtt{emb}_S(\emptyset^\sharp_S(c)) \rightarrow SKIP) \cap traces(\mathtt{emb}_T(\emptyset^\sharp_T(d)) \rightarrow SKIP) \end{array}$$

A simple example

- CSP-CASL specification:

$$\begin{array}{lll} \textbf{data} & \textbf{sorts} & S, T \\ & \textbf{ops} & c : S;\ d : T \\ \textbf{process} & & c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP \end{array}$$

- CSP-CASL semantics (for one model M of the data part):

$$\begin{array}{ll} & traces([\![c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP]\!]_\emptyset) \\ = & traces(\mathtt{emb}_S(\emptyset^\sharp_S(c)) \rightarrow SKIP \ \ || \ \ \mathtt{emb}_T(\emptyset^\sharp_T(d)) \rightarrow SKIP) \\ = & traces(\mathtt{emb}_S(\emptyset^\sharp_S(c)) \rightarrow SKIP) \cap traces(\mathtt{emb}_T(\emptyset^\sharp_T(d)) \rightarrow SKIP) \\ = & \{\langle\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c))\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c)), \checkmark\rangle\} \cap \\ & \{\langle\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d))\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d)), \checkmark\rangle\} \end{array}$$

- Question: $c \rightarrow SKIP \;\; || \;\; d \rightarrow SKIP \neq_{\mathcal{T}} Stop$?

- Question: $c \rightarrow SKIP \;\; || \;\; d \rightarrow SKIP \neq_{\mathcal{T}} Stop$?

$$\{\langle\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c))\rangle, \ldots\} \cap \{\langle\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d))\rangle, \ldots\} \neq \{\langle\rangle\}$$

- Question: $c \rightarrow SKIP \;\; || \;\; d \rightarrow SKIP \neq_{\mathcal{T}} Stop$?

$$\{\langle\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c))\rangle, \ldots\} \cap \{\langle\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d))\rangle, \ldots\} \neq \{\langle\rangle\}$$

iff

$$\mathtt{emb}_S(\emptyset^\sharp_S(c)) = \mathtt{emb}_T(\emptyset^\sharp_T(d))$$

- Question: $c \rightarrow SKIP \;\; || \;\; d \rightarrow SKIP \neq_{\mathcal{T}} Stop$?

$$\{\langle\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c))\rangle, \ldots\} \cap \{\langle\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d))\rangle, \ldots\} \neq \{\langle\rangle\}$$

iff

$$\mathtt{emb}_S(\emptyset^\sharp_S(c)) = \mathtt{emb}_T(\emptyset^\sharp_T(d))$$

iff (Expressing tests in FinCommSubPFOL$^=$)

$$\emptyset \mid\!\!\vdash_{\mathsf{FinCommSubPFOL}^=} c = d$$

- Question: $c \rightarrow SKIP \;\; || \;\; d \rightarrow SKIP \neq_{\mathcal{T}} Stop$?

$$\{\langle\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c))\rangle, \ldots\} \cap \{\langle\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d))\rangle, \ldots\} \neq \{\langle\rangle\}$$

iff

$$\mathtt{emb}_S(\emptyset^\sharp_S(c)) = \mathtt{emb}_T(\emptyset^\sharp_T(d))$$

iff (Expressing tests in FinCommSubPFOL$^=$)

$$\emptyset \mid\!\!\vdash_{\mathsf{FinCommSubPFOL}^=} c = d$$

iff (Representing FinCommSubPFOL$^=$ in CASL)

$$\emptyset \mid\!\!\vdash_{\mathrm{CASL}} \alpha(c = d)$$

- Question: $c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP \neq_{\mathcal{T}} Stop$?

$$\{\langle\rangle, \langle \mathtt{emb}_S(\emptyset^\sharp_S(c))\rangle, \ldots\} \cap \{\langle\rangle, \langle \mathtt{emb}_T(\emptyset^\sharp_T(d))\rangle, \ldots\} \neq \{\langle\rangle\}$$

iff

$$\mathtt{emb}_S(\emptyset^\sharp_S(c)) = \mathtt{emb}_T(\emptyset^\sharp_T(d))$$

iff (Expressing tests in FinCommSubPFOL$^=$)

$$\emptyset \mid\!\!\vdash_{\mathsf{FinCommSubPFOL}^=} c = d$$

iff (Representing FinCommSubPFOL$^=$ in CASL)

$$\emptyset \mid\!\!\vdash_{\mathrm{CASL}} \alpha(c = d)$$

iff

$$\emptyset \mid\!\!\vdash_{\mathrm{CASL}} F$$

Consequence: $c \rightarrow SKIP \ \ || \ \ d \rightarrow SKIP =_{\mathcal{T}} Stop$

Properties of CSP-CASL

1. Supports full CASL (basic & structured specifications)
   in the data part:
   - Loose & initial specification
   - Subsorting & partiality
   - Structuring
2. Works with all denotational CSP models such as $\mathcal{T}, N, F$.
3. Simple refinement notion with clear relations to
   - Data refinement and
   - Process refinement

Properties of CSP-CASL

1. Supports full CASL (basic & structured specifications) in the data part:
   - Loose & initial specification
   - Subsorting & partiality
   - Structuring
2. Works with all denotational CSP models such as $\mathcal{T}, N, F$.
3. Simple refinement notion with clear relations to
   - Data refinement and
   - Process refinement

See paper in TCS 2006.

Closely related approaches: the data part

- Initial specification in CEL (no partiality, no subsorting):
  - Lotos (Iso 8807, 1989)
  - CCS-CASL (Salaün et. al, 2001 & 2002)
- Functional language with fixed data types:
  - E-Lotos (2002)
  - CSP_M (input language of FDR)
- Loose specification in equational logic (no partiality, no subsorting):
  - μCRL (Groote, 1995)

Some reactive CASL extensions

Framework

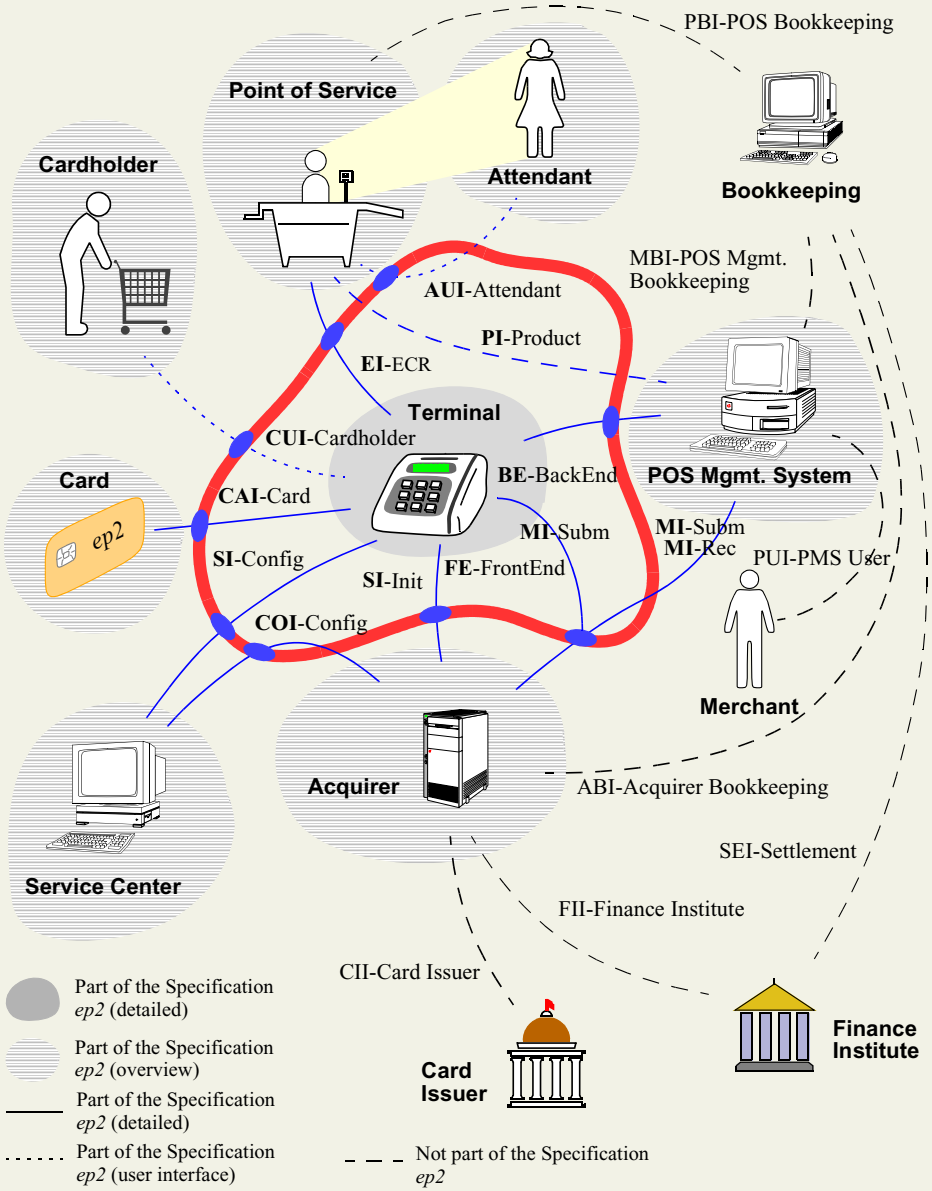
CSP-CASL
CCS-CASL
CASL-CHART
DO-CASL

Meta-Framework

CASL-LTL

CO-CASL

Modelling ep2 in CSP-CASL


PBI-POS Bookkeeping
Point of Service
Attendant
Bookkeeping
Cardholder
MBI-POS Mgmt.
Bookkeeping
AUI-Attendant
PI-Product
EI-ECR
Terminal
CUI-Cardholder
BE-BackEnd
POS Mgmt. System
Card
CAI-Card
ep2
MI-Subm
MI-Subm
MI-Rec
PUI-PMS User
SI-Config
SI-Init
FE-FrontEnd
COI-Config
Merchant
Acquirer
ABI-Acquirer Bookkeeping
SEI-Settlement
Service Center
FII-Finance Institute
CII-Card Issuer
Finance
Institute
Card
Issuer
Part of the Specification
ep2 (detailed)
Part of the Specification
ep2 (overview)
Part of the Specification
ep2 (detailed)
Part of the Specification
ep2 (user interface)
Not part of the Specification
ep2

The ep2 Consortium

Cornèr Bank Card Center, Credit Suisse / Swisscard aecs, Swiss Post, Telekurs Multipay AG, Telekurs Card Solutions AG, Diners Club Schweiz AG, JCB International Co. Ltd., Verband Elektronischer Zahlungsverkehr VEZ.

Some terminal manufacturers:

Epsys AG, CCV-CardPay AG jeronimo SA, CCS Card Solutions, Telekurs Card Solutions AG, ICP Paysys GmbH, C.A.R.U.S Retail Information Technology GmbH, Thales e-Transactions GmbH.

ep2 specification style

- plain English
- UML-like notation (use cases, activity diagrams, message sequence charts, class models, etc.)
- pictures
- tables
- lists
- file descriptions
- encoding rules

Shortcomings

ep2 specification style is state of the art . . .

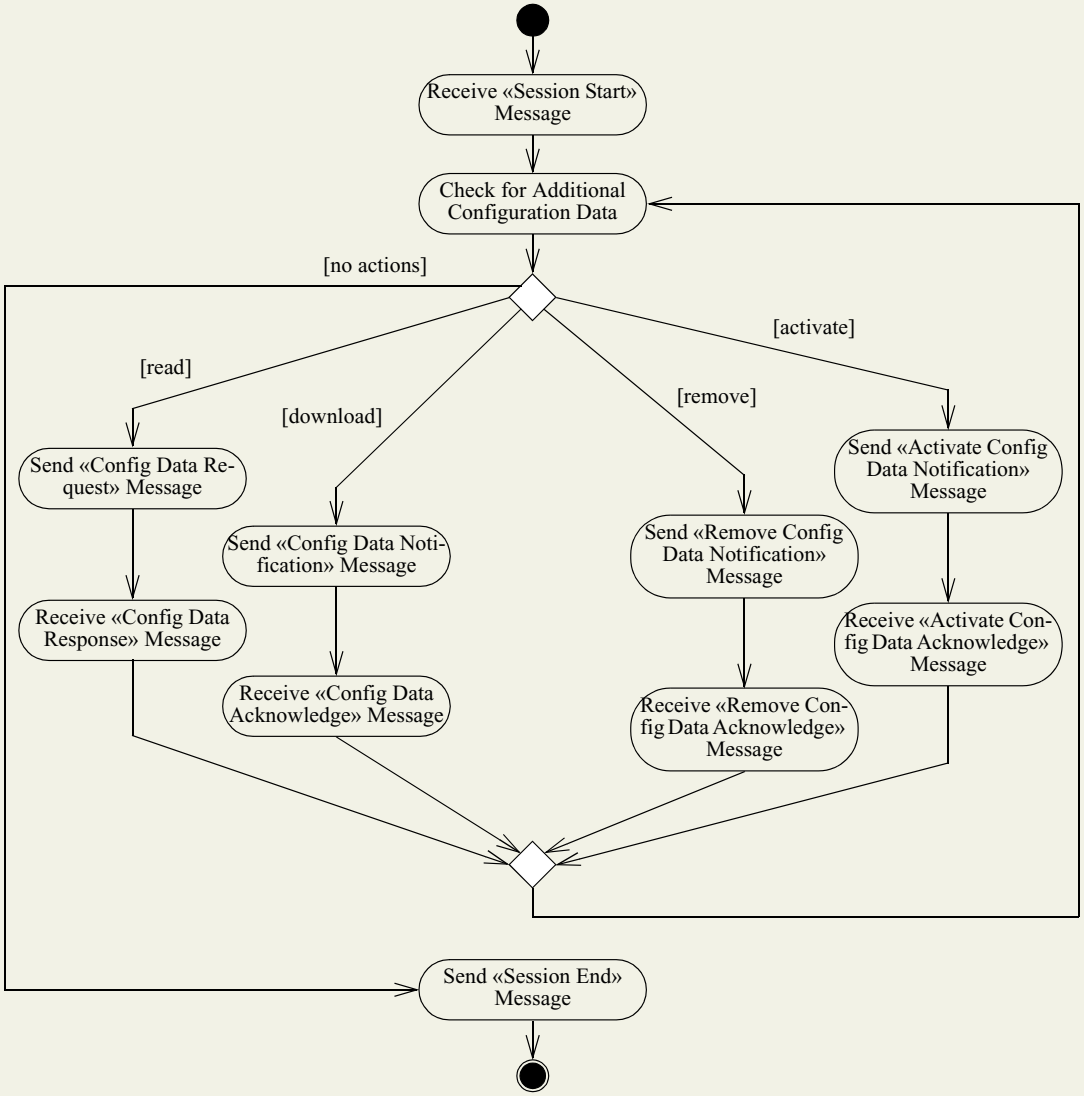
Shortcomings

ep2 specification style is state of the art . . .

. . . however, it is hard to produce a correct implementation:

- not usable for tool supported software development
- interleaving between different levels of abstraction
- ambiguities $\rightsquigarrow$ interoperability problems
- inconsistencies within/between documents

A typical EP2 dialog: the Acquirer's side



. . . formalised in CSP-CASL

```
AcquirerInit =    C_SI-Init ? start: D_SessionStart
               -> CManagement

CManagement =  ExitCManagement |~| ReadC |~| DownloadC ...

ExitCManagement = C_SI-Init ! end: D_SessionEnd -> Skip

ReadC =
     C_SI-Init ! request:  D_ConfigDataRequest
  -> C_SI-Init ? response: D_ConfigDataResponse
  -> CManagement
```

See paper at WADT 04.

CSP-Prover: Towards tool support for CSP-CASL

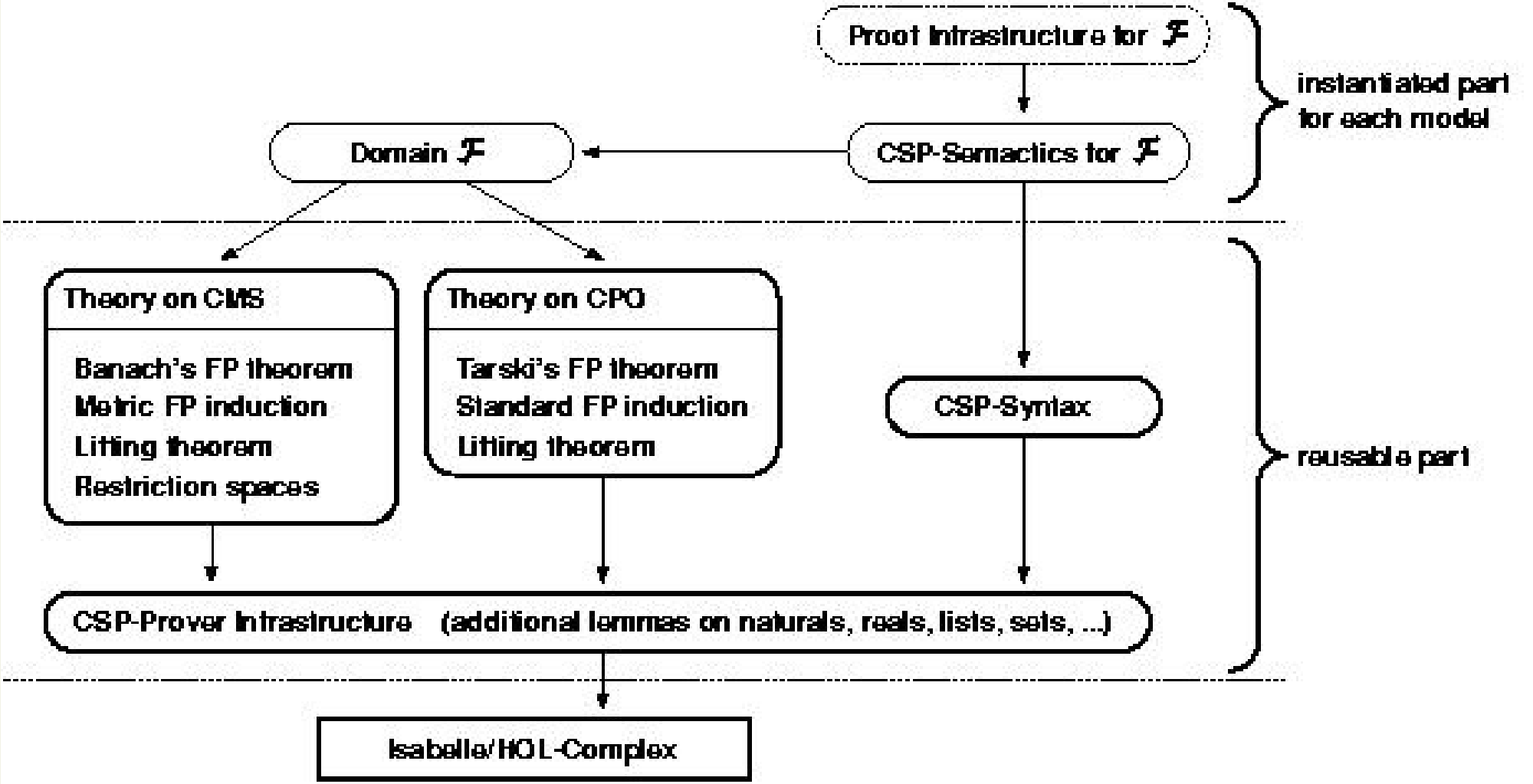
CSP-Prover

(cooperation with Y.Isobe (AIST, Japan))

Refinement proofs '$P \sqsubseteq Q$' in various CSP models

- Based on Isabelle-HOL (Complex)
- Generic architecture
- At the moment: stable-failures-model (and traces-model)
- Deep encoding
- Denotational CSP-semantics (metric, partial order)

Theory map


Proof Infrastructure for $\mathcal{F}$
Domain $\mathcal{F}$
CSP-Semantics for $\mathcal{F}$
instantiated part for each model
Theory on CMS
Banach's FP theorem
Metric FP induction
Lifting theorem
Restriction spaces
Theory on CPO
Tarski's FP theorem
Standard FP induction
Lifting theorem
CSP-Syntax
reusable part
CSP-Prover Infrastructure (additional lemmas on naturals, reals, lists, sets, ...)
Isabelle/HOL-Complex

Deadlocks in CSP

P is in a deadlock after trace s if

$$(s, A \cup \{\checkmark\}) \in failures(P)$$

Stable-failures refinement:

$$P \sqsubseteq_{\mathcal{F}} Q :\Leftrightarrow traces(P) \supseteq traces(Q) \wedge failures(P) \supseteq failures(Q)$$

Facts:

- Stable-failures abstraction preserves deadlocks:
  Q has a deadlock, $P \sqsubseteq_{\mathcal{F}} Q \Rightarrow$ P has a deadlock.
- Sequential processes (prefix, non-deterministic choice)
  are deadlock free.

Proof of deadlock-freedom in CSP-Prover

$$\texttt{Init} \sqsubseteq_{\mathcal{F}} \texttt{TerminalInit} \ \texttt{|[C\_SI\_Init]|} \ \texttt{AquirerInit}$$

where

```
Init = C_SI-Init ! start: D_SessionStart
       -> Loop

Loop =     C_SI-Init ! end: D_SessionEnd -> SKIP
       |~|    C_SI-Init ! request:  D_ConfigDataRequest
           -> C_SI-Init ! response: D_ConfigDataResponse
           -> Loop
       |~|    ...
```

Proof of deadlock-freedom in CSP-Prover

$$\texttt{Init} \sqsubseteq_{\mathcal{F}} \texttt{TerminalInit} \ \texttt{|[C\_SI\_Init]|} \ \texttt{AquirerInit}$$

where

```
Init = C_SI-Init ! start: D_SessionStart
       -> Loop

Loop =     C_SI-Init ! end: D_SessionEnd -> SKIP
       |~|    C_SI-Init ! request:  D_ConfigDataRequest
           -> C_SI-Init ! response: D_ConfigDataResponse
           -> Loop
       |~|    ...
```

See papers at TACAS '05, FOSE '05, CONCUR '06.

Perspective

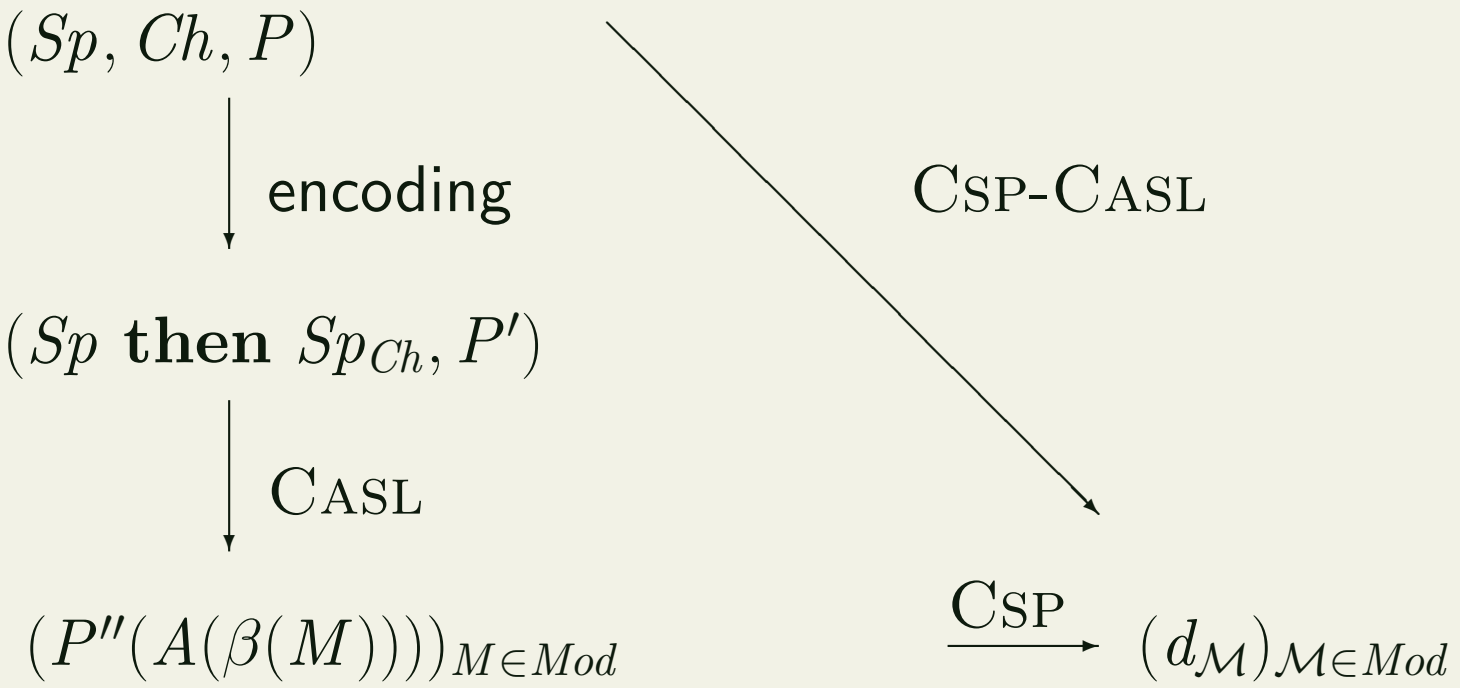
EPSRC project 'Processes and Data'

Starting date: 1.8.2006

- Topics:
  Structured CSP-CASL, Refinement, Tools, Testing.
- Doctoral students:
  A.Gimblett, L.O'Reilly, G.Samuel, T.Kahsai.
- Visiting researchers:
  Y.Isobe (Japan), T.Mossakowski (Bremen), H.Schlingloff (Berlin).
- Industrial Partners:
  Zühlke (Switzerland) – EP2 specification.
  C.A.R.U.S (Germany) – EP2 terminal.

Appendix

The semantical construction



$$Mod := \mathbf{Mod}(Sp\ \mathbf{then}\ Sp_{Ch})$$

The new aspects of CSP-CASL

Data:

'traditional' approach	CSP-CASL
initial semantics	loose + initial sem.
CEL	SubPCFOL (with local top elem.)

The new aspects of CSP-CASL

Data:

'traditional' approach	CSP-CASL
initial semantics	loose + initial sem.
CEL	SubPCFOL (with local top elem.)

Processes:

operational	generic (denotational)

The new aspects of CSP-CASL

Data:

'traditional' approach	CSP-CASL
initial semantics	loose + initial sem.
CEL	SubPCFOL (with local top elem.)

Processes:

operational	generic (denotational)

Proof tools:

model checking	aim: theorem proving