Formal Modeling, Verification and Refinement of Long Running Transactions

Ji Wang

National Laboratory for Parallel and Distributed Processing, Changsha, China

Joint work with Zhenbang Chen and Zhiming Liu
Agenda

- Background and motivation
- Compensating CSP (cCSP)
- Non-determinism and deadlock
- Livelock and refinement
- Algebraic laws
- Conclusion and next step
Long-Running Transactions

- Database
  - Long-lived transactions
  - Small ACID transactions

- SAGAS
  - 1987, SIGMOD

- Compensation
Compensation
In Database

- An activity has its compensation activity
- In case of a failure, use compensations
- Atomicity and consistency

Error -> Failure
In Distributed Computing

World wide distributed organizations

Coordinate to accomplish a task

Long Running Transactions

How to ensure consistency in case of a failure?
Orchestration Programming in SOC

WS-BPEL

Compensation based fault handling

Flexible recovery mechanisms for LRTs

Ensure an acceptable consistency of composite Web Services

WS-BPEL 2.0, OASIS Standard, 11 April 2007
Orchestration Programming in SOC

- **WS-BPEL**
  - Compensation based fault handling
  - Flexible recovery mechanisms for LRTs

- **Formal languages**
  - cCSP, StAC, SAGAs Calculi, etc.
Formal Modeling and Verification

Modeling
- Rigorous semantic foundation
- Formal semantics for industrial languages
- Basis for verification

Verification
- Ensure the correctness of LRTs
- Improve the reliability of LRT designs
Compensating CSP (cCSP)

Compensating CSP (cCSP)

- Process language
  - CSP extension for modeling LRTs
  - Basic operators
- Two types of processes
  - Standard & Compensable
- Terminated trace semantics
cCSP Syntax and Example

\[ P ::= a \mid P; P \mid P \boxdot P \mid P \parallel P \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{throw} \mid \text{yield} \]

\[ PP ::= P \div P \mid PP; PP \mid PP \boxdot PP \mid PP \parallel PP \mid PP \triangleright PP \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \]

Example \[ [(a_1 \div b_1; a_2 \div b_2) \ ; \ \text{throww}] \]
cCSP Syntax and Example

\[
P ::= a \mid P; P \mid P\square P \mid P\parallel P \mid P\triangleright P \mid [PP] \mid \text{skip} \mid \text{throw} \mid \text{yield}
\]

\[
PP ::= P;PP \mid PP\square PP \mid PP\parallel PP \mid PP\triangleright PP \mid [PP] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd}
\]

Example \[ [(a_1 \div b_1; a_2 \div b_2) ; \text{throww}] \]
cCSP Syntax and Example

Example: \([(a_1 \div b_1; a_2 \div b_2) ; \text{throww}]\)

\[
P ::= a | P; P | P \parallel P | P \mid P | P \triangleright P | [PP] | \text{skip} | \text{throw} | \text{yield}
\]

\[
PP ::= P \div P | PP; PP | PP \parallel PP | PP \mid PP | PP \triangleright PP | \text{skipp} | \text{throww} | \text{yieldd}
\]
cCSP Syntax and Example

\[
P ::= a \mid P; P \mid P\parallel P \mid P\mid P \mid P\triangleright P \mid [PP] \mid \text{skip} \mid \text{throw} \mid \text{yield}
\]

\[
PP ::= \boxed{P\parallel P} \mid PP; PP \mid PP\parallel PP \mid PP\mid PP \mid PP\triangleright PP \mid \text{skipp} \mid \text{throww} \mid \text{yieldd}
\]

Example \[\[(a_1\parallel b_1;a_2\parallel b_2) \ ; \ \text{throww}\]\]
cCSP Syntax and Example

\[
P ::= a \mid P; P \mid P\sqcap P \mid P\parallel P \mid P\triangleright P \mid [PP] \mid \text{skip} \mid \text{throw} \mid \text{yield}
\]

\[
PP ::= P\sqcup P \mid PP; PP \mid PP\sqcap PP \mid PP\parallel PP \mid PP\triangleright PP \mid [PP] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd}
\]

Example \[ [(a_1\div b_1; a_2\div b_2) ; \text{throww}] \]

\[
\begin{align*}
a_1 & \quad a_2 \quad \odot \quad b_2 \\
b_1 &
\end{align*}
\]
cCSP Syntax and Example

\[
P ::= a | P; P | P \sqcap P | P \parallel P | P \triangleright P | [PP] | \text{skip} | \text{throw} | \text{yield}
\]

\[
PP ::= P \div P | PP; PP | PP \sqcap PP | PP \parallel PP | PP \triangleright PP | \text{skipp} | \text{throww} | \text{yieldd}
\]

Example \[ (a_1 \div b_1; a_2 \div b_2) ; \text{throww} \]

\[
a_1 \quad a_2 \quad \text{☹} \quad b_2 \quad b_1
\]
Terminated Trace Semantics

\[ T(a) =_{\text{def}} \{a, \sqrt{\}} \]
\[ T(\text{skip}) =_{\text{def}} \{\sqrt{\} \} \quad T(\text{throw}) =_{\text{def}} \{!\} \quad T(\text{yield}) =_{\text{def}} \{\sqrt{\}, ?\} \]
\[ T(P;Q) =_{\text{def}} \{s_1 ; s_2 | s_1 \in T(P), s_2 \in T(Q)\} \]
\[ T(P\parallel Q) =_{\text{def}} \{s | \exists s_1 \in T(P), s_2 \in T(Q), s \in s_1 \parallel s_2\} \]
\[ T(P \triangleright Q) =_{\text{def}} \{s_1 \triangleright s_2 | s_1 \in T(P), s_2 \in T(Q)\} \]
Examples

\[ P ::= \quad a \mid P; P \mid P \parallel P \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{throw} \mid \text{yield} \]

\[ PP ::= \quad P; P \mid PP \parallel PP \mid PP \psi PP \mid PP \parallel PP \mid PP \triangleright PP \mid [PP] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \]

\[
T(a) = \{<a, \sqrt{\rangle}>\}
\]
\[
T(a;b) = \{<a, b, \sqrt{\rangle}>\} \quad T(a;\text{throw};b) = \{<a, !>\}
\]
\[
T(a \parallel b) = \{<a, b, \sqrt{\rangle}, <b, a, \sqrt{\rangle}>\}
\]
\[
T((a;\text{throw}) \parallel b) = \{<a, b, !>, <b, a, !>\}
\]
\[
T((a;\text{throw}) \parallel (\text{yield};b)) = \{<a, !>, <b, a, !>, <a, b, !>\}
\]
\[
T(a \triangleright b) = \{<a, \sqrt{\rangle}>\} \quad T((a;\text{throw}) \triangleright b) = \{<a, b, \sqrt{\rangle}>\}
\]
Terminated Trace Semantics

\[ T(P ÷ Q) \] =_{def} \{ s_1 ÷ s_2 \mid s_1 \in T(P), s_2 \in T(Q) \}

if \( s_1 = t^\sqrt{\cdot} \), \( s_1 ÷ s_2 = (s_1, s_2) \), else \( (s_1, <\sqrt{\cdot}> ) \)

\[ T(\text{skipp}) \] =_{def} \text{skip} ÷ \text{skip}

\[ T(\text{throww}) \] =_{def} \text{throw} ÷ \text{skip}

\[ T(\text{yieldd}) \] =_{def} \text{yield} ÷ \text{skip}

Examples

\[ T(a ÷ b) = \{(a, !>, b, \sqrt{\cdot})\} \]

\[ T((a; \text{throw}) ÷ b) = \{(a, !>, <\sqrt{\cdot}>\}) \]
Terminated Trace Semantics

\[
T(PP;Q) = \text{def} \ \{(p, p') ; (q, q') \mid (p, p') \in T(\langle P \rangle), (q, q') \in T(\langle Q \rangle)\}
\]

if \( \text{p} = t^\sqrt{\text{\_}}, \) \( (p, p') ; (q, q') = (p; q, q'; p'), \)
else, \( (p, p') ; (q, q') = (p, p'), \)

\[
T(a_1 \div b_1; a_2 \div b_2) = \{(a_1, a_2, \sqrt{\_}), (b_2, b_1, \sqrt{\_})\}
\]
Terminated Trace Semantics

\[ T([PP]) = \{ s_1 \hat{\times} s_2 \mid (s_1 \hat{!}, s_2) \in T(PP) \} \cup \{ s_1 \hat{\checkmark} \mid (s_1 \hat{\checkmark}, s_2) \in T(PP) \} \]

Examples

\[ T(a \div b) = \{ (a, \checkmark), (b, \checkmark) \} \quad T([a \div b]) = \{ a, \checkmark \} \]
\[ T(a \div b; \text{throww}) = \{ (a, !), (b, \checkmark) \} \quad T([a \div b; \text{throww}]) = \{ a, b, \checkmark \} \]
### Semantics Example

<table>
<thead>
<tr>
<th>$P$</th>
<th>::=</th>
<th>$a$</th>
<th>$P$ ; $P$</th>
<th>$P \sqcap P$</th>
<th>$P \parallel P$</th>
<th>$P \triangleright P$</th>
<th>$[PP]$</th>
<th>skip</th>
<th>throw</th>
<th>yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PP$</td>
<td>::=</td>
<td>$P$ ; $P$</td>
<td>$PP ; PP$</td>
<td>$PP \sqcap PP$</td>
<td>$PP \parallel PP$</td>
<td>$PP \triangleright PP$</td>
<td>skipp</td>
<td>throww</td>
<td>yieldd</td>
<td></td>
</tr>
</tbody>
</table>

**Example**  
$[(a_1 \div b_1 ; a_2 \div b_2) ; \text{throww}]$

**Trace Semantics:** $\{ <a_1, a_2, b_2, b_1, \sqrt{\text{}} > \}$
Theoretical Issues of cCSP

**Concurrent systems**

- Non-determinism & Deadlock & Livelock

**Reason**

- Trace semantics, no synchronization, no recursion

**Refinement**
Non-determinism and Deadlock

Zhenbang Chen and Zhiming Liu. An Extended cCSP with Stable Failures Semantics. 7th International Colloquium on Theoretical Aspects of Computing (ICTAC’10), LNCS 6255, 2010.
Non-determinism & Deadlock

- Extend the syntax of cCSP
- Internal and external choices
- Synchronization, hiding and renaming

\[
P ::= a | P;P | P \land P | P \lor P | P | P \mid X | P[R] | P > P | [PP] |
\]

\[
PP ::= P;P | PP;PP | PP \land PP | PP \lor PP | PP | PP \mid X |
\]

skip, stop, throw, yield

\[
PP[R] | skipp | throww | yieldd
\]
Non-determinism & Deadlock

- Extend the syntax of cCSP
- Internal and external choices
- Synchronization, hiding and renaming
- A stable failures semantics
- Use refusals to model deadlocks
Basic Idea of a Failure-based Semantics

A failure \( (s, X) \)

One trace \( s \) that a process can execute

The set of the events that the process refuses to perform after executing \( s \)

\[
\begin{align*}
\{<\cdot, X) | a \notin X \} & \cup \{<a>, X) | b \notin X \} & \cup \{<a, b>, X) | \exists \notin X \} & \cup \{<a,b,\exists>, X) | X \subseteq \Sigma \}
\end{align*}
\]

A process **deadlocks** if it refuses to perform any event after executing a **non-terminated** trace.
Semantic Models

- **Standard processes**
  \[ [P] = (T, F) \]
  - Trace set, \( T_s(P) \)
  - Failure set, \( F_s(P) \)

- **Compensable processes**
  \[ [PP] = (T, F, C) \]
  - Forward Trace set, \( T_c(PP) \)
  - Forward Failure set, \( F_c(PP) \)
  - Compensation Set, \( C(PP), (s, T, F) \)
Semantics (1)

\[
P ::= a | P;P | P \cap P | P \square P | P \parallel P | P \setminus X | P[R] | P \triangleright P | [PP] | \text{skip} | \text{stop} | \text{throw} | \text{yield}
\]

\[
PP ::= P \downarrow P | PP;PP | PP \cap PP | PP \square PP | PP \parallel PP | PP \setminus X | PP[R] | \text{skipp} | \text{throww} | \text{yieldd}
\]

\[T_s(a)=\{\langle\rangle, \langle a\rangle, \langle a, \sqrt{\rangle}\}\]

\[F_s(a)=\{\langle\rangle, X\mid a \notin X\} \cup \{(\langle a\rangle, X) \mid \sqrt{\notin X}\} \cup \{(\langle a, \sqrt{\rangle}, X)\}
\]

where \(X \subseteq \Sigma \cup \{!,?,\sqrt{\}\}\)

\[T_s(\text{stop})=\{\langle\rangle\}\]

\[F_s(\text{stop})=\{\langle\rangle, X\mid X \subseteq \Sigma \cup \{!,?,\sqrt{\}\}\}\]
Semantics (2) – Internal and External Choices

\[ P ::= a \mid P;P \mid P \sqcap P \mid P \sqcup P \mid P \parallel P \mid P \setminus X \mid P[R] \mid P > P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \]

\[ PP ::= P \div P \mid PP;PP \mid PP \sqcap PP \mid PP \sqcup PP \mid PP \parallel PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \]

Difference is at the beginning

1. Internal choice will refuse an event if any sub process can refuse it
2. External choice will refuse an event if both sub processes can refuse it
Semantics (2) – Internal and External Choices

\[
P ::= a \mid P;P \mid P \sqcap P \mid P \sqcup P \mid P \parallel P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \\
skip \mid \text{stop} \mid \text{throw} \mid \text{yield}
\]

\[
PP ::= P \div P \mid PP;PP \mid PP \sqcap PP \mid PP \sqcup PP \mid PP \parallel PP \mid PP \setminus X \mid \\
PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd}
\]

\[
T_s(P \sqcap Q) = T_s(P) \cup T_s(Q) \quad T_s(P \sqcup Q) = T_s(P) \cup T_s(Q)
\]

\[
F_s(P \sqcap Q) = F_s(P) \cup F_s(Q)
\]

\[
F_s(P \sqcup Q) = \{(\llcorner, X) \mid (\llcorner, X) \in F_s(P) \cap F_s(Q)\} \quad ...
\]
Semantics (2) – Internal and External Choices

\[ \Sigma = \{a, b\} \]

\[ F_s(a \sqcap b) = \{(<> , X) | X \subseteq \{b, !, ?, \sqrt{\ }\}\} \cup \{(<> , X) | X \subseteq \{a, !, ?, \sqrt{\ }\}\} \ldots \]

\[ F_s(a \Box b) = \{(<> , X) | X \subseteq \{!, ?, \sqrt{\ }\}\} \ldots \]

\[ T_s(a \Box b) = \{<> , <a>, <a, \sqrt{\ }>, <b>, <b, \sqrt{\ }\} = T_s(a \sqcap b) \]
Semantics (3) – Synchronization

1. Parallel composition with synchronization on $X$ can refuse an event out of $X$ if both sub processes can refuse it.
2. Parallel composition with synchronization on $X$ can refuse an event in $X$ if any sub process can refuse it.
Semantics (3) – Synchronization

\[
P ::= a \mid P;P \mid P \sqcap P \mid P \diamond P \mid P \parallel P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield}
\]

\[
PP ::= P \div P \mid PP;PP \mid PP \sqcap PP \mid PP \diamond PP \mid PP \parallel PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd}
\]

\[
F_s(P \parallel Q) = \{(s, X_1 \cup X_2) \mid \exists (s_1, X_1) \in F_s(P), (s_2, X_2) \in F_s(Q), X_1 \setminus (X \cup W) = X_2 \setminus (X \cup W) \land s \in s_1 \parallel X \setminus s_2 \}
\]

where \( W = \{!, ?, \sqrt{ }\} \)
Semantics (3) - How to have a deadlock

No synchronization, no deadlock

\[
F_s(a) = \{ (\langle\rangle,X) \mid a \notin X \} \cup \{ (\langle a \rangle, X) \mid \checkmark \notin X \} \cup \{ (\langle a,\checkmark \rangle, X) \}
\]

\[
F_s(b) = \{ (\langle\rangle,X) \mid b \notin X \} \cup \{ (\langle b \rangle, X) \mid \checkmark \notin X \} \cup \{ (\langle b,\checkmark \rangle, X) \}
\]

\[
F_s(a \parallel b) \text{ is } \{ (\langle\rangle,X) \mid X \subseteq \{a,b,!,?,\checkmark\}\}, \text{ i.e. } F_s(\text{stop})\{a,b\}
\]
Semantics (4)

\[
P ::= a | P;P | P \sqcap P | P \sqcap P | P \parallel P | P | P \backslash X | P[R] | P > P | [PP] | \text{skip} | \text{stop} | \text{throw} | \text{yield}
\]

\[
PP ::= [P : P] | PP;PP | PP \sqcap PP | PP \sqcap PP | PP \parallel PP | PP | PP \backslash X | PP[R] | \text{skipp} | \text{throww} | \text{yieldd}
\]

\[
\begin{align*}
[ a \div b ] &= (T_s(a), F_s(a), \{ (<a, \sqrt{\top}, T_s(b), F_s(b)) \}) \\
C((a \sqcap (a;\text{throw})) \div b) &= \{ (<a, \sqrt{\top}, T_s(b), F_s(b)) , \\
&\quad (<a, !\top, T_s(\text{skip}), F_s(\text{skip})) \} \\
[[a \div b]] &= (T_s(a), F_s(a))
\end{align*}
\]
Semantic (5)

$P ::= a | P;P | P \cap P | P \Box P | P \parallel P | P \setminus X | P[R] | P \triangleright P | [PP] |
skip \mid stop \mid throw \mid yield$

$PP ::= P;PP | PP;PP | PP \cap PP | PP \Box PP | PP \parallel PP | PP \setminus X | PP[R] | skipp \mid throww \mid yieldd$

$[[PP \cap QQ]] =_{\text{def}} (T_1 \cup T_2, F_1 \cup F_2, C_1 \cup C_2)$

$[[PP \Box QQ]] =_{\text{def}} (T_s(PP_f \Box QQ_f), F_s(PP_f \Box QQ_f), C_1 \cup C_2)$

Trace set function of $P$

Failure set function of $P$
Semantics (5) – Example

\[ P ::= a \mid P;P \mid P \cap P \mid P \sqcup P \mid P \parallel P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid \lbrack PP \rbrack \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \]

\[ PP ::= P \div P \mid PP;PP \mid \lbrack PP \cap PP \rbrack \mid PP \sqcup PP \mid PP \parallel PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \]

Example \[ [(a_1 \div b_1 \cap a_2 \div b_2) \mid \text{throww}] \]

\[ (a_1 ; b_1) \cap (a_2 ; b_2) \]
Semantics (6)

\[\begin{align*}
P & ::= \ a \mid P;P \mid P \sqcap P \mid P \square P \mid P \ || \ P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \\
PP & ::= \ P \div P \mid PP;PP \mid PP \sqcap PP \mid PP \square PP \mid PP \ || PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd}
\end{align*}\]

\[\begin{align*}
[PP;QQ] & =_{\text{def}} (T_s(PP_f;QQ_f), F_s(PP_f;QQ_f), C) \\
C & =_{\text{def}} \{(s, T, F) \mid \exists (s_1, PP_c) \in C(PP), (s_2, QQ_c) \in C(QQ), (s_1=t^\top \land s=t^\top s_2 \land T=T_s(QQ_c ; PP_c) \land F=F_s(QQ_c ; PP_c)) \lor (s_1\neq t^\top \land s=s_1 \land T=T_s(PP_c) \land F=F_s(PP_c))\}\]
\]
Semantics (6) – Example

Example \[ [(a_1 \div b_1; a_2 \div b_2) ; \text{throww}] \]

\[ a_1 \quad a_2 \quad \frown \quad b_2 \quad b_1 \quad a_1 \quad a_2 \quad b_2 \quad b_1 \]
Semantics (6) – Example

\[
P ::= a | P ; P | P \sqcap P | P \sqcap P | P \sqcup P | P \parallel P | P \setminus X | P[R] | P > P | [PP] | \text{skip} | \text{stop} | \text{throw} | \text{yield}
\]

\[
PP ::= P \sqcup P | [PP]; PP | PP \sqcap PP | PP \sqcup PP | PP \parallel PP | PP \setminus X | PP[R] | \text{skipp} | \text{throww} | \text{yieldd}
\]

Example \([(a_1 \div b_1; a_2 \div b_2) ; \text{throww}]\)

\[
\frac{a_1}{b_1} \quad \frac{a_2}{b_2} \quad \text{☹} \quad b_2 \quad b_1 \quad a_1 ; a_2 ; b_2 ; b_1
\]

\[
\llbracket a_1 \div b_1; a_2 \div b_2 \rrbracket = \langle T_s(a_1;a_2), F_s(a_1;a_2) \rangle_{36}, \{(a_1,a_2,\rightarrow), T_s(b_2;b_1), F_s(b_2;b_1)\}\}
\]
Semantics (7)

\[ P ::= a \mid P ; P \mid P \sqcap P \mid P \sqcup P \mid P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \]

\[ PP ::= P \div P \mid PP ; PP \mid PP \sqcap PP \mid PP \sqcup PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \]

\[
\llbracket PP \| QQ \rrbracket =_{\text{def}} (T_s(PP_f \| QQ_f), F_s(PP_f \| QQ_f), C)
\]

\[ C =_{\text{def}} \{(s, T, F) \mid \exists (s_1, PP_c) \in C(PP), (s_2, QQ_c) \in C(QQ), s \in (s_1 \| s_2) \wedge T = T_s(PP_c \| QQ_c) \wedge F = F_s(PP_c \| QQ_c)\} \]
Semantics (7) – Example

\[
P ::= a | P;P | P \land P | P|P | P \land X | P[R] | P > P | [PP] |
\]
\[
\text{skip} | \text{stop} | \text{throw} | \text{yield}
\]

\[
PP ::= P;P | PP;PP | PP \land PP | PP|PP | PP \land X | PP[R] | \text{skipp} | \text{throww} | \text{yieldd}
\]

Example  
\[
[(a_1 \lor b_1 \parallel a_2 \lor b_2)]
\]
\[
\text{Deadlock!!}
\]

Semantics: \{(<> , X) | X \subseteq \Sigma\}
Semantics (7) – Example

\[ P ::= a \mid P;P \mid P \sqcap P \mid P \sqcup P \mid P \parallel P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \]

\[ PP ::= P \sqcap P \mid PP;PP \mid PP \sqcap PP \mid PP \sqcup PP \mid PP \parallel PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \]

Example: \[ [(a \div b_1 \parallel a \div b_2) ; \text{throww}] \]
Summary Until Now

- An extension to cCSP
- Non-determinism
- Synchronized parallel composition
- A new semantic model for the extended cCSP
- Non-determinism and deadlock modeling

Zhenbang Chen and Zhiming Liu. An Extended cCSP with Stable Failures Semantics. 7th International Colloquium on Theoretical Aspects of Computing (ICTAC’10), LNCS 6255, 2010.
Livelock and Refinement


Livelock & Refinement

- No recursion
  - Cannot model divergence, i.e. livelock
  - Hard for a denotational semantics

- Refinement is hard to define w.r.t. the stable failures model
  - Design by refinement for LRTs
Livellock & Refinement

- Extend language
- Recursive processes
- Speculative choice

\[ P ::= a \ | \ P;P \ | \ P \ominus P \ | \ P \square P \ | \ P \parallel P \ | \ P \backslash X \ | \ P[R] \ | \ P \triangleright P \ | \ [PP] \ | \ \text{skip} \ | \ \text{stop} \ | \ \text{throw} \ | \ \text{yield} \ | \ \mu p.F(p) \]

\[ PP ::= P \diamond P \ | \ PP;PP \ | \ PP \ominus PP \ | \ PP \square PP \ | \ PP \parallel PP \ | \ PP \backslash X \ | \ PP[R] \ | \ skipp \ | \ throww \ | \ yieldd \ | \ \mu pp.FF(pp) \]
Livelock & Refinement

- Extend language
- Recursive processes
- Speculative choice
- A failure-divergence semantics
- Support recursion interpretation
- Use recursions to model livelocks
- Refinement definition
Basic Idea of a Failure-Divergence Semantics

A failure is $(s, X)$

- One trace $s$ that a process can execute
- The set of the events that the process refuses to perform after executing $s$

A divergence is a trace $s$

1. The process enters a chaos state after executing $s$
2. The process is totally unpredictable, i.e. it can perform or refuse any event
3. Use $\text{DIV}$ to denote the process that diverges immediately
Basic Idea of a Failure-Divergence Semantics

- A divergence is suffix closed

\[ s \in D(P) \cap \Sigma^* \Rightarrow s^t \in D(P) \]

- A divergent process can refuse any event

\[ s \in D(P) \Rightarrow (s, X) \in F(P), \text{ where } X \subseteq \Sigma \cup \{!, ?, \sqrt{\cdot}\} \]

- A terminated divergence must be generated by a non-terminated divergence

\[ s^w \in D(P) \Rightarrow s \in D(P), \text{ where } w \subseteq \{!, ?, \sqrt{\cdot}\} \]
Standard Processes

- Semantic model
  \[ [P] = (F, D) \]
  - Failure set, \( F(P) \)
  - Divergence set, \( D(P) \)

- Examples for divergence sets
  - \( D(a) = \{\} \)
  - \( D(DIV) \) contains \( <> \), i.e. \( D(DIV) \) contains any traces

- Refinement of standard processes
  - \( P_1 \sqsubseteq P_2 \stackrel{\text{def}}{=} F_1 \supseteq F_2 \land D_1 \supseteq D_2 \)
How to have a livelock

\[ P ::= a \mid P;P \mid P \sqcap P \mid P \circledast P \mid P \| P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \mid [\mu p. F(p)] \]

\[ PP ::= P \setminus P \mid PP;PP \mid PP \sqcap PP \mid PP \circledast PP \mid PP \| PP \mid PP \setminus PP \mid PP \setminus X \mid PP[R] \mid skipp \mid throww \mid yieldd \mid [\mu pp. F\!\!\!\!\(p\)](pp) \]

No recursion, no divergence

(\(\mu p. (a \; ; \; p)\)) executes a infinitely

(\(\mu p. (a \; ; \; p)\) \(\setminus \{a\}\) is equal to DIV
Semantic Models

- **Standard processes**

\[ [P] = (F(P), D(P)) \]

- **Failure set**, \( F(P) \)

- **Divergence set**, \( D(P) \)

- **Compensable processes**

\[
PP ::= P \div P \mid PP;PP \mid PP \sqcap PP \mid PP \bowtie PP \mid PP \parallel PP \mid PP \times PP \mid PP \setminus X \mid PP[R] \mid skipp \mid throww \mid yieldd \mid \mu pp.FF(pp)
\]

\[ [PP] = ??? \]
Ways to Go

Based on an existing one
Stable failures model

Build a new model
Find it out
Problems

We failed on the first way

Compensable processes

\[
\lbrack PP \rbrack = (T, F, C)
\]

\[
\lbrack PP \rbrack = (F, D, C)
\]

Extension

\[
(s, T, F)
\]

\[
(s, F, D)
\]

Complete lattice or CPO?

Refinement order?
Working Process and Final Result

- Search and tradeoff
- Semantic model and algebraic laws
- Refinement and fixed-point theory

\[
(F, D, F^c, D^c) \rightarrow (s, s', X) \rightarrow (s, s')
\]
Order and Properties

\[(F_1, D_1, F^c_1, D^c_1) \sqsubseteq_c (F_2, D_2, F^c_2, D^c_2)\]

- \(F_1 \supseteq F_2\)
- \(D_1 \supseteq D_2\)
- \(F^c_1 \supseteq F^c_2\)
- \(D^c_1 \supseteq D^c_2\)

- The order is easy to understand
- The domain is a CPO w.r.t. the order
- The order is natural for refinement
Semantic Models

Standard processes

\[ [P] = (F, D) \]

Failure set, \( F(P) \)

Divergence set, \( D(P) \)

Compensable processes

\[ [PP] = (F, D, F^c, D^c) \]

Forward Failure set, \( F_f(PP) \)

Forward Divergence set, \( D_f(PP) \)
Semantic Models

- **Standard processes**
  \[ [P] = (F, D) \]
  - Failure set, \( F(P) \)
  - Divergence set, \( D(P) \)

- **Compensable processes**
  \[ [[PP]] = (F, D, F^c, D^c) \]
  - Compensation Failure set, \( F^c(PP) \)
  - Compensation Divergence set, \( D^c(PP) \)
Semantics (1)

\[
\begin{align*}
P ::= & \ a \mid P;P \mid P \sqcap P \mid P \sqcup P \mid P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \\
& \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \mid \mu p. F(p) \\
PP ::= & \ P \bowtie P \mid PP;PP \mid PP \sqcap PP \mid PP \sqcup PP \mid PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \mid \mu pp. FF(pp)
\end{align*}
\]

\[
\begin{align*}
\llbracket a \div b \rrbracket &= (F(a), \{\}, \{<a,\sqrt{\_}>\}\times F(b), \{\}) \\
F^c((a \sqcap (a; \text{throw})) \div b) &= \{<a,\sqrt{\_}>\}\times F(b) \cup \{<a,\_!>\}\times F(\text{skip})
\end{align*}
\]
Semantics (2)

The operators are continuous

Least fixed-point semantics

\[
\left[ \mu pp. FF(pp) \right] = \bigsqcup \{ FF^n(DIV \div DIV) \mid n \in \mathbb{N} \}
\]
Semantics (2) – Example

P ::= a | P;P | P □ P | P □ P | P || P | P \ X | P[R] | P ⊷ P | [PP] |
skip | stop | throw | yield | µ p.F(p)

PP ::= P ÷ P | PP;PP | PP □ PP | PP □ PP | PP || PP | PP \ X | PP[R] | skipp | throww | yieldd | µ pp.FF(pp)

Examples

[ µ pp.(a ÷ b;pp) ; throww ]

a a a a a a a a a a a a ... Not terminated
b b b b b b b b b b ... Not terminated

[[ µ pp.(a ÷ b;pp) ] = ( [ µ p.(a;p) ] ,{},{} )]
**Semantics (3)**

\[
P ::= \ a \mid P;P \mid P \bowtie P \mid P \parallel P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \\
\text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \mid \mu p.F(p)
\]

\[
PP ::= \ P \div P \mid PP;PP \mid PP \bowtie PP \mid PP \parallel PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \mid \mu pp.FF(pp)
\]

**Examples**

\[
[(a_1 \div b_1 \bowtie (a_2 \div b_2; \text{throww})); \text{throww}]
\]

\[
(a_1 \parallel a_2; \text{throw} \quad b_2 \quad \frown \quad b_1 \quad (a_1 \parallel a_2); b_2; b_1)
\]
Semantics (3)

\[
P ::= a | P;P | P \sqcap P | P \sqcup P | P \parallel X | P \setminus X | P[R] | P \triangleright P | [PP] | skip \sqcap stop | throw \sqcap yield | \mu p.F(p)
\]

\[
PP ::= P\sqcup P | PP;PP | PP \sqcap PP | PP \sqcup PP | PP \parallel X | PP \setminus X | PP[R] | skipp | throww | yeildd | \mu pp.FF(pp)
\]

Examples

\[
[(a_1 \sqcup b_1 \times a_2 \sqcup b_2); throww]
\]
Semantics (3)

\[ P ::= a \mid P;P \mid P \Box P \mid P \parallel P \mid P \dot{\boxtimes} X \mid P[R] \mid P \supset P \mid \{PP\} \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \mid \mu p.F(p) \]

\[ PP ::= P \div P \mid PP;PP \mid PP \Box PP \mid PP \parallel PP \mid PP \dot{\boxtimes} X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \mid \mu pp.FF(pp) \]

Examples
\[
(((a_1 \div b_1; \text{throww}) \Box (a_2 \div b_2; \text{throww})) \parallel (a_3 \div b_3))
\]
Semantics (3)

\[
P ::= a \mid P;P \mid P \sqcap P \mid P \boxdot P \mid P \sqcup P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \mid \mu p.F(p)
\]

\[
PP ::= P \div P \mid PP;PP \mid PP \sqcap PP \mid PP \boxdot PP \mid PP \sqcup PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \mid \mu pp.MM(pp)
\]

Examples

\[\left[\left(\left(a_1 \div b_1;\text{throww}\right) \boxdot \left(a_2 \div b_2;\text{throww}\right)\right) \parallel \left(a_3 \div b_3\right)\right]\]

\[\begin{align*}
  a_1 & ; \text{throw} \parallel a_2 \div \text{throw} \\
  b_1 & \quad b_2
\end{align*}\]
Semantics (3)

\[ P ::= a \mid P;P \mid P \sqcap P \mid P \sqcup P \mid P \mid P \setminus X \mid P[X] \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \mid \mu p.F(p) \]

\[ PP ::= P \div P \mid PP;PP \mid PP \sqcap PP \mid PP \sqcup PP \mid PP \mid PP \mid PP \setminus X \mid PP[X] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \mid \mu pp.FF(pp) \]

Examples

\[ (((a_1 \div b_1; \text{throww}) \triangleright (a_2 \div b_2; \text{throww})) \mid (a_3 \div b_3)) \]

\( (a_1 \parallel a_2) ; \text{throw} \)

\( b_1 \quad b_2 \)
Semantics (3)

\[ P ::= a \mid P;P \mid P \sqcap P \mid P \setminus P \mid P \parallel P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid skip \mid stop \mid throw \mid yield \mid \mu p.F(p) \]

\[ PP ::= P\div P \mid PP;PP \mid PP \sqcap PP \mid PP \setminus PP \mid PP \parallel PP \mid PP \setminus X \mid PP[R] \mid skipp \mid throww \mid yieldd \mid \mu pp.FF(pp) \]

Examples

\[ (((a_1 \div b_1; \text{throww}) \times (a_2 \div b_2; \text{throww})) || (a_3 \div b_3)) \]

\[ (a_1 || a_2) ; \text{throw} \mid || a_3 \]

\[ b_1 \mid b_2 \mid b_3 \]
Semantics (3)

\[
P ::= a \mid P ; P \mid P \cap P \mid P \Box P \mid P \parallel P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \\
\text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \mid \mu p.F(p)
\]

\[
PP ::= P ; P \mid PP ; PP \mid PP \cap PP \mid PP \Box PP \mid PP \parallel PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \mid \mu pp.FF(pp)
\]

Examples

\[
(((a_1 \div b_1 ; \text{throww}) \Box (a_2 \div b_2 ; \text{throww})) \parallel (a_3 \div b_3))
\]

(a_1 \parallel a_2 \parallel a_3)
Semantics (3)

\[ P ::= a \mid P;P \mid P \sqcap P \mid P \sqcap P \mid P \parallel X \mid P[R] \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \mid \mu p.F(p) \]

\[ PP ::= P \sqcup P \mid PP;PP \mid PP \sqcap PP \mid PP \sqcap PP \mid PP \parallel PP \mid PP \parallel PP \mid PP \parallel [PP \times PP] \mid PP \backslash X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \mid \mu pp.FF(pp) \]

Examples

\[ (((a_1 \div b_1; \text{throww}) \times (a_2 \div b_2; \text{throww})) \parallel (a_3 \div b_3)) \]

\[ (a_1 \parallel a_2 \parallel a_3) \]

\[ \text{unhappy} \quad b_1 \parallel b_2 \parallel b_3 \]
Semantics (3)

\[ P ::= a \mid P;P \mid P \sqcap P \mid P \Box P \mid P \parallel P \mid P \setminus X \mid P[R] \mid P \triangleright P \mid [PP] \mid \text{skip} \mid \text{stop} \mid \text{throw} \mid \text{yield} \mid \mu p.F(p) \]

\[ PP ::= P \triangleright P \mid PP;PP \mid PP \sqcap PP \mid PP \Box PP \mid PP \parallel PP \mid PP \setminus X \mid PP[R] \mid \text{skipp} \mid \text{throww} \mid \text{yieldd} \mid \mu pp.FF(pp) \]

Examples

\[ (((a_1 \triangleright b_1; \text{throww}) \Box (a_2 \triangleright b_2; \text{throww})) \parallel (a_3 \triangleright b_3)) \]

(a_1 \parallel a_2 \parallel a_3)
Livelock & Refinement

- All basic concurrent features
  - Divergence for livelock
- A failure-divergence semantics
  - Fixed-point theory
- Refinement w.r.t the semantics
  - Non-determinism

Algebraic Laws

Algebraic Laws of Standard Processes
Some Still Valid CSP laws

- **Idempotence**
  \[ P \sqcap P = P \]
  \[ P \sqcap P = P \]

- **Units and zeros**
  \[ \text{skip} ; P = P \]
  \[ \text{stop} \square P = P \]
  \[ P \setminus \{\} = P \]
  \[ \text{stop} ; P = \text{stop} \]
  \[ \text{DIV} \sqcap P = \text{DIV} \]

- **Refinement**
  \[ P \sqcap Q \subseteq P \]
  \[ \text{DIV} \subseteq P \]
Exception Handling

Units and zeros

- throw \(\triangleright P = P\)
- \(P \triangleright \text{throw} = P\)
- skip \(\triangleright P = \text{skip}\)

Distribution and association

- \(P \triangleright (Q \cap R) = (P \triangleright Q) \cap (P \triangleright R)\)
- \((P \cap Q) \triangleright R = (P \triangleright R) \cap (Q \triangleright R)\)
- \(P \triangleright (Q \triangleright R) = (P \triangleright Q) \triangleright R\)
Parallel Composition

- **Unit and zeros**
  - \( \text{throw} \parallel \text{skip} = \text{throw} \)
  - \( \text{skip} \parallel \text{throw} = \text{throw} \)
  - \( \text{throw} \parallel \text{skip} = \text{throw} \)
  - \( \text{P} \parallel \text{skip} = \text{P} \)

- **If \( \text{P} \) does not terminate with a yield terminal event**
  - \( \text{throw} \parallel \text{P} = \text{P} \parallel \text{throw} \)
  - \( \text{throw} \parallel (\text{yield} ; \text{P}) = \text{throw} \cap (\text{P} ; \text{throw}) \)
Algebraic Laws of Compensable Processes
Basic Algebraic Laws

Units and zeros

\[ \text{skipp} \; PP = PP \]
\[ PP \; \text{skipp} = PP \]
\[ \text{throww} \; PP = \text{throww} \]

Distribution

\[ [PP \cap QQ] = [PP] \cap [QQ] \]
\[ P \div (Q \cap R) = (P \div Q) \cap (P \div R) \]
\[ (P \div Q) \setminus X = (P \setminus X) \div (Q \setminus X) \]
Refinement Laws

\[ PP \sqcap QQ \subseteq_c PP \]

Consistently related

\[ PP_1 \subseteq_c PP_2 \Rightarrow [PP_1] \subseteq [PP_2] \]

Reduction

\[ Q_1 \subseteq Q_2 \Rightarrow P \div Q_1 \subseteq_c P \div Q_2 \]
\[ P_1 \subseteq P_2 \Rightarrow P_1 \div Q \subseteq_c P_2 \div Q \]
Compensation Laws (1)

If $P_1$ and $P_2$ do not result in an exception

$[P_1 \div Q_1 ; \text{throww}] = P_1 ; Q_1$

$[P_1 \div Q_1 ; P_2 \div Q_2 ; \text{throww}] = P_1 ; P_2 ; Q_2 ; Q_1$

The laws are still valid when $P_1$ is YIELD
Compensation Laws (2)

If all the standard processes terminate successfully and do not diverge

\[[P \div Q] \parallel \text{throww} = P ; Q\]

\[P_1 \div Q_1 \parallel P_2 \div Q_2 = P_1 \parallel P_2 \div Q_1 \parallel Q_2\]

\[[P_1 \div Q_1 \boxtimes P_2 \div Q_2) ; \text{throww} = (P_1 \parallel P_2) ; ((Q_1 ; Q_2) \sqcap (Q_2 ; Q_1))\]
Interruption Laws

If all the standard processes do not diverge and terminate successfully

\[(\text{yieldd;}P_1 ÷ Q_1;\text{yieldd;}P_2 ÷ Q_2) \parallel \text{throww} = \text{skip} ∩ (P_1 ; Q_1) ∩ (P_1 ; P_2 ; Q_2 ; Q_1)\]

\[(\text{yieldd;}P_1 ÷ Q_1) \parallel (\text{yieldd;}P_2 ÷ Q_2) \parallel \text{throww} = \text{skip} ∩ (P_1 ; Q_1) ∩ (P_2 ; Q_2) ∩ ((P_1 \parallel P_2);(Q_1 \parallel Q_2))\]

\text{yieldd} must be used to specify interruption places
Conclusion & Ongoing Work

- A semantic theory for LRTs
  - Non-determinism, deadlock and livelock
  - Design by refinement
  - Reasoning LRTs by algebraic laws

- Ongoing work
  - PAT based model checker for extended cCSP
  - Application of the theory, e.g., BPMN
End
Thank you!