An Extended cCSP with Stable Failures Semantics

Zhenbang Chen\textsuperscript{1} \quad Zhiming Liu\textsuperscript{2}

National Laboratory for Parallel and Distributed Processing
Changsha, China

International Institute for Software Technology, United Nations University
Macau, China

September 1, 2010
Outline

- Background and Motivation
- Introduction to cCSP
- Extended cCSP and Stable Failures Semantics
  - Standard Process
  - Compensable Process
- Conclusion
Outline

- Background and Motivation
- Introduction to cCSP
- Extended cCSP and Stable Failures Semantics
  - Standard Process
  - Compensable Process
- Conclusion
LRT is from the transaction processing in database area
   - Some database operations last for a long time, but also need to ensure to be a transaction
   - Use compensation to handle failures

LRT models attract attention recently because of the progress in Service-Oriented Computing (SOC)
   - Coordination between wide-spread communicating peers
   - Atomic transaction is too strict for this scenario
   - LRT can tackle this problem by using compensation

Some modeling and programming languages for LRT
   - Industrial: WS-BPEL, XLANG, BPMN
   - Formal: SAGAS, StAC, cCSP
A theoretical foundation for LRT modeling

What we need
- Denotational model
- Ensure the laws of LRT
- Compositional reasoning

Problems
- Most formal languages for LRT only have an operational semantics
  - SAGAS, StAC
- cCSP is an exception
  - A trace semantics is provided
Modeling LRT with cCSP

- cCSP is a variant of CSP
  - Standard process
  - Compensable process
- Limitation
  - Only an operational semantics beside the trace semantics
  - The operators are limited
    - No parallel composition with synchronization
    - No non-deterministic choice
Outline

- Background and Motivation
- Introduction to cCSP
- Extended cCSP and Stable Failures Semantics
  - Standard Process
  - Compensable Process
- Conclusion
Recovery in cCSP

- Same as the backward recovery of SAGAS
Syntax of cCSP

\[
P ::= A \\
| P ; P \\
| P \parallel P \\
| SKIP \\
| THROW \\
| YIELD \\
| P \triangleright P \\
| [PP]
\]

\[
PP ::= P \div P \\
| PP ; PP \\
| PP \parallel PP \\
| PP \square PP \\
| PP \oplus PP \\
| SKIP \\
| THROW \\
| YIELD \\
| YIELDD \\
| [PP]
\]
Terminating trace semantics

- Terminal event set $\Omega = \{\checkmark, !, ?\}$
- The semantic model is a set of terminating traces

<table>
<thead>
<tr>
<th>Atomic Action</th>
<th>For all $A \in \Sigma$, $T(A) = {\langle A, \checkmark \rangle}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Process</td>
<td>$T(YIELD) = {\langle ? \rangle, \langle \checkmark \rangle}$</td>
</tr>
<tr>
<td>Parallel Composition</td>
<td>$p \hat{\langle} \omega_1 \hat{\rangle} \parallel q \hat{\langle} \omega_2 \hat{\rangle} = {r \hat{\langle} \omega_1 &amp; \omega_2 \hat{\rangle}</td>
</tr>
<tr>
<td></td>
<td>$T(P \parallel Q) = {r</td>
</tr>
</tbody>
</table>

where

<table>
<thead>
<tr>
<th>$\omega_1$</th>
<th>$\checkmark$</th>
<th>$\checkmark$</th>
<th>$\checkmark$</th>
<th>$!$</th>
<th>$!$</th>
<th>$?$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega_2$</td>
<td>$\checkmark$</td>
<td>$?$</td>
<td>$!$</td>
<td>$!$</td>
<td>$?$</td>
<td>$?$</td>
</tr>
<tr>
<td>$\omega_1 &amp; \omega_2$</td>
<td>$\checkmark$</td>
<td>$?$</td>
<td>$!$</td>
<td>$!$</td>
<td>$!$</td>
<td>$?$</td>
</tr>
</tbody>
</table>
Trace Semantics of Compensable Process

Terminating trace semantics

- The semantic model is a set of terminating trace pairs

**Compensation Pair**

\[
p \div q = \begin{cases} 
(p, q) & p = p_1 \langle \checkmark \rangle \\
(p, \checkmark) & p = p_1 \langle \omega \rangle \land \omega \neq \checkmark 
\end{cases}
\]

\[
T_c(P \div Q) = \{p \div q \mid p \in T(P) \land q \in T(Q)\} \cup \{(\langle ? \rangle, \langle \checkmark \rangle)\}
\]

**Sequential Composition**

\[
(p, p') ; (q, q') = \begin{cases} 
(p_1 \langle q, q' \rangle ; p') & p = p_1 \langle \checkmark \rangle \\
(p, p') & p = p_1 \langle \omega \rangle \land \omega \neq \checkmark 
\end{cases}
\]

\[
T_c(PP ; QQ) = \{(p, p') ; (q, q') \mid (p, p') \in T_c(PP) \land (q, q') \in T_c(QQ)\}
\]

**Transaction Block**

\[
\mathcal{T}(\llbracket PP \rrbracket) = \{p \hat{p}' \mid (p \langle ! \rangle, p') \in T_c(PP)\} \cup \{p \langle \checkmark \rangle \mid (p \langle \checkmark \rangle, p') \in T_c(PP)\}
\]
The following laws do not hold:

\[ PP ; SKIPP = PP \]

\[ \mathcal{T}_c(A \div B ; SKIPP) = \{(\langle A, \checkmark \rangle, \langle B, \checkmark \rangle), (\langle A, ? \rangle, \langle B, \checkmark \rangle), (\langle ?, \checkmark \rangle)\} \]

\[ \mathcal{T}_c(A \div B) = \{(\langle A, \checkmark \rangle, \langle B, \checkmark \rangle), (\langle ?, \checkmark \rangle)\} \]

\[ [P \div P'] = P \text{ if } P \text{ is non-yielding} \]

\[ \mathcal{T}([THROW \div A]) = \{\langle \checkmark \rangle\} = \mathcal{T}(SKIP) \neq \mathcal{T}(THROW) \]

\[ [P \div P'; THROWW] = P ; P' \text{ if } P \text{ is non-yielding} \]

The reason is the same as that of the above one.
Problems and Discussion (2)

- Some useful operators are not provided
  - Non-deterministic choice
  - Parallel composition with synchronization
  - Hiding
  - Renaming

- These operators are important
  - Deadlock behavior
  - Modeling systems at different levels
Our Contribution

We extend and modify cCSP as follows

- Distinguish choice operators
  - Internal (□) and External (□) Choice
- Introduce new operators
  - Synchronized parallel composition, Hiding, Renaming
- A stable failures semantics for both standard and compensable processes
  - Deadlock, distinguish choices
  - No implicit interruption
- Fix the problems pointed out before
Outline

- Background and Motivation
- Introduction to cCSP
- Extended cCSP and Stable Failures Semantics
  - Standard Process
  - Compensable Process
- Conclusion
Syntax of the Extended cCSP

\[
P ::= \begin{align*}
| & A & | & PP \\
| & P ; P & | & PP ; PP \\
| & P \cap P & | & PP \cap PP \\
| & P \boxdot P & | & PP \boxdot PP \\
| & P \parallel X & | & PP \parallel PP \\
| & \text{SKIP} & | & \text{SKIPP} \\
| & \text{THROW} & | & \text{THROWW} \\
| & \text{YIELD} & | & \text{YIELDD} \\
| & \text{STOP} & | & PP \setminus X \\
| & P \setminus X & | & PP[R] \\
| & P[R] & | & [PP] \\
| & P \triangleright P & |
\end{align*}
\]
We use traces and the events that a process refuses to perform to give the semantics of the process.

- For the process $A; B$
  - At the beginning, the process refuses to execute any event except $A$
  - After performing $A$, the process refuses to execute any event except $B$
  - After executing the trace $\langle A, B \rangle$, the process needs to terminate, so it will refuse any event except $✓$
  - Finally, the process terminates, it refuses to perform any event

- If a deadlock occurs, processes will refuse to perform any event.
Outline

- Background and Motivation
- Introduction to cCSP
- Extended cCSP and Stable Failures Semantics
  - Standard Process
  - Compensable Process
- Conclusion
Semantic Domain of the Standard Process

- Stable failures semantics \((T, F)\)

- Semantic functions
  - Trace set function: \(T_S(P): \mathcal{P} \rightarrow \mathbb{P}(\Sigma^*\Omega)\)
  - Failure set function: \(F_S(P): \mathcal{P} \rightarrow \mathbb{P}(\Sigma^*\Omega \times \mathbb{P}(\Sigma^\Omega))\)

- Axioms of the semantic domain:
  \[
  \begin{align*}
  & T \text{ is non-empty and prefix closed} & (1) \\
  & (s, X) \in F \Rightarrow s \in T & (2) \\
  & (s, X) \in F \land Y \subseteq X \Rightarrow (s, Y) \in F & (3) \\
  & (s, X) \in F \land \forall a \in Y \cdot s^\langle a \rangle \not\in T \Rightarrow (s, X \cup Y) \in F & (4) \\
  & s^\langle \omega \rangle \in T \Rightarrow (s, \Sigma^\Omega \setminus \{\omega\}) \in F, \text{ where } \omega \in \Omega & (5) \\
  & s^\langle \omega \rangle \in T \Rightarrow (s^\langle \omega \rangle, X) \in F, \text{ where } \omega \in \Omega \land X \subseteq \Sigma^\Omega & (6)
  \end{align*}
\]
Trace synchronization, where \( s_1, t_1 \in \Sigma^* \) and \( \omega, \omega_1, \omega_2 \in \Omega \)

\[
\begin{align*}
& s_1 \parallel X t_1 \langle \omega \rangle = \{\} \\
& s_1 \langle \omega_1 \rangle \parallel X t_1 \langle \omega_2 \rangle = \{ u \langle \omega_1 \& \omega_2 \rangle \mid u \in s_1 \parallel X t_1 \}
\end{align*}
\]

Semantic definition

\[
\begin{align*}
\mathcal{T}_S(P \parallel X Q) &= \{ u \mid \exists s \in \mathcal{T}_S(P), t \in \mathcal{T}_S(Q) \cdot u \in (s \parallel X t) \} \\
\mathcal{F}_S(P \parallel X Q) &= \{ (u, E) \mid (u, E) \in (s, Y) \oplus (t, Z) \land \exists s, t \cdot (s, Y) \in \mathcal{F}_S(P) \land (t, Z) \in \mathcal{F}_S(Q) \}
\end{align*}
\]

Laws need to hold:

\[
\begin{align*}
\text{THROW} \parallel X \text{SKIP} &= \text{YIELD} & \text{THROW} \parallel X \text{YIELD} &= \text{THROW} \\
\text{YIELD} \parallel X \text{SKIP} &= \text{THROW}
\end{align*}
\]
Consider the parallel execution of $P$ and $Q$

- $P \parallel_X Q$ can refuse an event in $X \cup \Omega$ if either $P$ or $Q$ can refuse it.
- $P \parallel_X Q$ can refuse an event outside $X \cup \Omega$ only if both $P$ and $Q$ can refuse it.

However, we need to take into account the synchronization between terminal events.

- $P \parallel_X Q$ cannot terminate if either $P$ or $Q$ cannot terminate.
- $P \parallel_X Q$ can terminate if both $P$ and $Q$ can terminate, and the synchronized terminal event should be removed from the refusal set of the synchronized failure.
First example, $\Sigma = \{A, B\}$ and $A \parallel (B \; THROW)$

- $A$ has the failure $(\langle\rangle, \{B, \checkmark, !, ?\})$
- $B \; THROW$ has the failure $(\langle B\rangle, \{B, \checkmark, ?\})$
- The synchronized failure set is $\{(\langle B\rangle, \{B, \checkmark, !, ?\})\}$
First example, $\Sigma = \{A, B\}$ and $A \parallel (B; THROW) \{\}$

- $A$ has the failure $((\), \{B, \checkmark, !, ?\})$
- $B; THROW$ has the failure $((B), \{B, \checkmark, ?\})$
- The synchronized failure set is $\{((B), \{B, \checkmark, !, ?\})\}$

Second example, $\Sigma = \{A\}$ and $A \parallel (A; THROW) \{A\}$

- $A$ has the failure $((A), \{A, !, ?\})$
- $A; THROW$ has the failure $((A), \{A, \checkmark, ?\})$
- Both processes can terminate after executing $(A)$, and the synchronized terminal event is $!$
- The synchronized failure set is $\{((A), \{A, \checkmark, ?\})\}$
Failure synchronization, where \((s, Y) \in \mathcal{F}_S(P)\) and \((t, Z) \in \mathcal{F}_S(Q)\)

\[(s, Y) \oplus (t, Z) = \begin{cases} 
(u, Y \cup Z) & | \ Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \land u \in s \parallel t \\
\text{if } (s, Y \cup \Omega) \in \mathcal{F}_S(P) \lor (t, Z \cup \Omega) \in \mathcal{F}_S(Q) \\
((u, (Y \cup Z) \setminus \Theta) & | \ Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \land \\
& u \in s \parallel t \land \Theta = rf(\omega_1, \omega_2)) \\
\text{otherwise}
\end{cases}\]

\(\omega_1\) is the terminal event \(P\) can perform to terminate
\[\forall (s, Y_1) \in \mathcal{F}_S(P) \bullet Y \subseteq Y_1 \Rightarrow (\omega_1 \in \Omega \land \omega_1 \notin Y_1)\]

\(\omega_2\) is the terminal event \(Q\) can perform to terminate
\[\forall (t, Z_1) \in \mathcal{F}_S(Q) \bullet Z \subseteq Z_1 \Rightarrow (\omega_2 \in \Omega \land \omega_2 \notin Z_1)\]

\(rf\) is the function for synchronizing \(\omega_1\) and \(\omega_2\)
In some cases, \( \omega_1 \) or \( \omega_2 \) may not exist, such as \((\langle \rangle, \{?\})\) of the process \( SKIP \sqcap THROW \).

If \( \omega_1 \) or \( \omega_2 \) does not exist, we use \( \epsilon \) to represent it.

\[
rf(\omega_1, \omega_2) = \begin{cases} 
\{\omega_1 \& \omega_2\} & \text{if } \omega_1 \in \Omega \land \omega_2 \in \Omega \\
\{\omega_1\} & \text{if } \omega_1 \in \Omega \land \omega_2 = \epsilon \\
\{\omega_2\} & \text{if } \omega_1 = \epsilon \land \omega_2 \in \Omega \\
\emptyset & \text{if } \omega_1 = \epsilon \land \omega_2 = \epsilon
\end{cases}
\]

More laws for parallel composition

\[
THROW \parallel P = P ; THROW \\
THROW \parallel (YIELD ; P) = THROW \sqcap (P ; THROW)
\]
Outline

1. Background and Motivation
2. Introduction to cCSP
3. Extended cCSP and Stable Failures Semantics
   - Standard Process
   - Compensable Process
4. Conclusion
The behavior of a compensable process
  - Forward behavior and compensation behavior

The compensation behavior needs to be recorded during the execution of the forward behavior
  - Maintain the relation between forward behavior and its compensation
  - Record the compensation behavior in right sequence

The semantic model of a compensable process $PP$ is a triple $(T, F, C)$
  - $T$ is the trace set of the forward behavior of $PP$
  - $F$ is the failure set of the forward behavior of $PP$
  - $C$ is the compensation behavior set, and each element is a $(s, T^c, F^c)$
The semantics of $PP$ can be calculated by the semantics functions as $(T^c(PP), F^c(PP), C(PP))$

- The forward trace set function
  \[ T^c(PP) : \mathcal{PP} \rightarrow \mathcal{P}({\Sigma^*}{\Omega}), \]
- The forward failure set function
  \[ F^c(PP) : \mathcal{PP} \rightarrow \mathcal{P}({\Sigma^*}{\Omega} \times \mathcal{P}(\Sigma{\Omega})), \]
- The compensation behavior set function
  \[ C(PP) : \mathcal{PP} \rightarrow \mathcal{P}({\Sigma^*}_{\Omega} \times \mathcal{P}({\Sigma^*}{\Omega}) \times \mathcal{P}({\Sigma^*}{\Omega} \times \mathcal{P}(\Sigma{\Omega}))). \]

For a compensable process $PP$ whose semantics is $(T, F, C)$, we use $PP_f$ to denote $(T, F)$.

For an element $(s, T^c, F^c)$ in $C$, we use $PP_c$ to denote $(T^c, F^c)$.

We will use the operators of standard processes for $PP_f$ and $PP_c$ as if there are standard processes.
If $P$ terminates successfully, process $Q$ will be recorded for compensating the effects caused by $P$ to recover from the failure that may happen in the future.

Semantic definition

\[ \mathcal{T}^c(P \div Q) = \mathcal{T}_S(P) \]
\[ \mathcal{F}^c(P \div Q) = \mathcal{F}_S(P) \]
\[ \mathcal{C}(P \div Q) = \{(s, F^c, D^c) | \exists s \in (\mathcal{T}_S(P) \cap \Sigma^*_\Omega) \bullet (s = t^\langle \checkmark \rangle \land T^c = \mathcal{T}_S(Q) \land F^c = \mathcal{F}_S(Q)) \lor (s \in \Sigma^*_{\{!,?\}} \land T^c = \mathcal{T}_S(SKIP) \land F^c = \mathcal{F}_S(SKIP))\} \]
If an exception occurs in the forward behavior of PP, the compensation behavior will executed.

\[ T_S([PP]) = (T^c(PP) \cap \Sigma^*\{\checkmark,?\}) \cup \{s_1 \mid \exists(s,T^c,F^c) \in C(PP) \bullet s = t^{\langle!\rangle} \land s_2 \in T^c \land s_1 = t^{\hat{s}_2}\} \]

\[ F_S([PP]) = \{(s,X) \mid s \in \Sigma^* \land (s,X \cup \{!\}) \in F^c(PP)\} \cup \{(s_1,X_1) \mid \exists(s,T^c,F^c) \in C(PP) \bullet \]
\[ (s \in \Sigma^*\{\checkmark,?\} \land s_1 = s \land X_1 \subseteq \Sigma^\Omega) \lor \]
\[ (s = t^{\langle!\rangle} \land (s_2,X_2) \in F^c \land s_1 = t^{\hat{s}_2} \land X_1 = X_2)\} \]

Law

\[ [P \div Q] = P \triangleright SKIP \]
The forward parts will be composed sequentially

$$\mathcal{T}_S(PP \ ; \ QQ) = \mathcal{T}_S(P_P^f \ ; \ Q_Q^f) \quad \mathcal{F}_c^c(PP \ ; \ QQ) = \mathcal{F}_S(P_P^f \ ; \ Q_Q^f)$$

The compensation parts will be composed in the reversed order

$$\mathcal{C}(PP \ ; \ QQ) = \{(s, T^c, F^c) \mid \exists (s_1, PP^c) \in \mathcal{C}(PP), (s_2, QQ^c) \in \mathcal{C}(QQ) \bullet (s_1 = t \langle \checkmark \rangle \wedge s = t \langle s_2 \rangle \wedge T^c = \mathcal{T}_S(Q_Q^c \ ; \ P_P^c) \wedge F^c = \mathcal{F}_S(Q_Q^c \ ; \ P_P^c)) \lor (s_1 \neq t \langle \checkmark \rangle \wedge s = s_1 \wedge T^c = \mathcal{T}_S(P_P^c) \wedge F^c = \mathcal{F}_S(P_P^c))\}$$
Sequential Composition:

$$PP ; SKIPP = PP$$

If $$P_i$$ and $$Q_i$$ ($$i \in \{1, 2\}$$) will not terminate with an exception

$$[P_1 \div Q_1 ; THROWW] = P_1 ; Q_1$$
$$[P_1 \div Q_1 ; P_2 \div Q_2 ; THROWW] = P_1 ; P_2 ; Q_2 ; Q_1$$

Parallel Composition:

If $$P_i$$ and $$Q_i$$ ($$i \in \{1, 2\}$$) will terminate successfully

$$[(P_1 \div Q_1 \parallel P_2 \div Q_2); THROWW] = (P_1 \parallel P_2) ; (Q_1 \parallel Q_2)$$
$$[(P_1 \div Q_1 ; P_2 \div Q_2) \parallel THROWW] = P_1 ; P_2 ; Q_2 ; Q_1$$
$$[(YIELD ; P_1 \div Q_1 ; YIELD ; P_2 \div Q_2) \parallel THROWW] = SKIP \sqcap (P_1 ; Q_1) \sqcap (P_1 ; P_2 ; Q_2 ; Q_1)$$
$$[(YIELD ; P_1 \div Q_1) \parallel (YIELD ; P_2 \div Q_2) \parallel THROWW] = SKIP \sqcap (P_1 ; Q_1) \sqcap (P_2 ; Q_2) \sqcap (P_1 \parallel P_2) ; (Q_1 \parallel Q_2)$$
Comparision with Original cCSP

- New and often used operators are introduced both in standard and compensable processes
- The semantic mode incorporates refusal information
- Unlike cCSP, we do not allow implicit interruption in compensable process
  - The designer of a LRT should specify the place where the LRT can be interrupted
- Yield interrupting behavior is kept in the semantics of the transaction block
  - Relax the assumptions of some laws
Outline

- Background and Motivation
- Introduction to cCSP
- Extended cCSP and Stable Failures Semantics
  - Standard Process
  - Compensable Process
- Conclusion
We extend the cCSP

- Distinguish the internal and external choices
- Introduce new operators: synchronized parallel composition, hiding, and renaming

A new semantics model for the extended cCSP

New laws for the extended cCSP
Future Work

- Stable failures semantics for recursive compensable process
- Failure divergence semantics for extended cCSP
- Refinement theory for LRT
- Axiomatic system and theorem proving tool for extended cCSP
Thank you very much!

Questions?