An Extended cCSP with Stable Failures Semantics

Zhenbang Chen[1] Zhiming Liu[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

Background - Long Running Transaction (LRT)

- 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

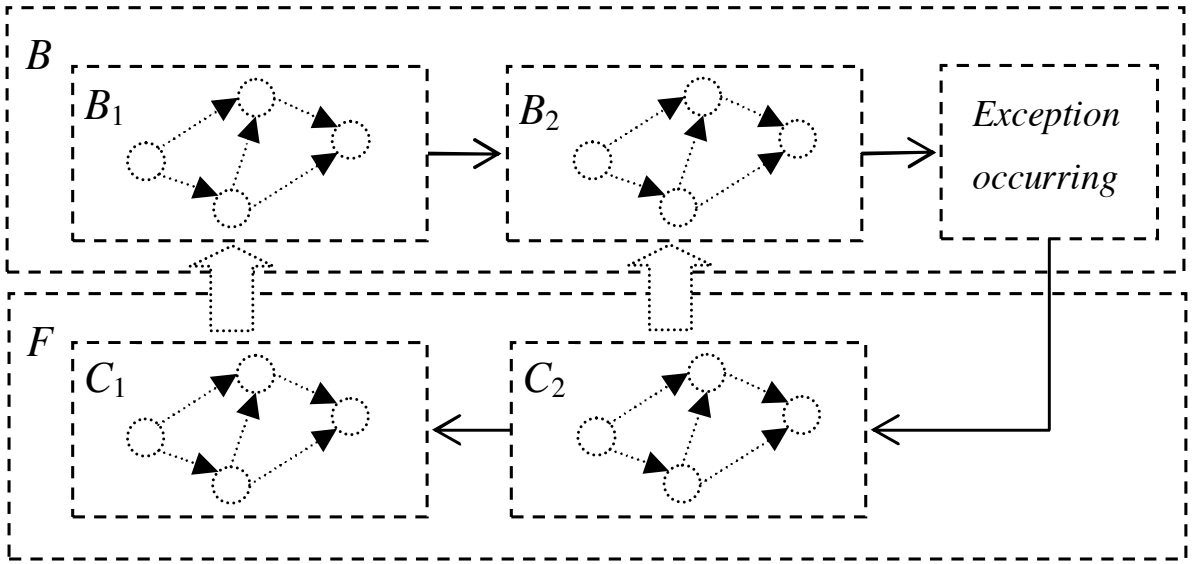
- cCSP is a variant of CSP
  - Standard process
  - Compensable process
- Limitation
  - Only an operational semantics beside the trace semantics
  - The operators are limited
    - No parallel composition with synchronization
    - No non-deterministic choice

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

$$
\begin{array}{lcl}
P & ::= & A \\
 & | & P \ ; \ P \\
 & | & P \Box P \\
 & | & P \parallel P \\
 & | & SKIP \\
 & | & THROW \\
 & | & YIELD \\
 & | & P \rhd P \\
 & | & [PP]
\end{array}
\qquad
\begin{array}{lcl}
PP & ::= & P \div P \\
 & | & PP \ ; \ PP \\
 & | & PP \Box PP \\
 & | & PP \parallel PP \\
 & | & SKIPP \\
 & | & THROWW \\
 & | & YIELDD
\end{array}
$$

Trace Semantics of Standard Process

Terminating trace semantics

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

Atomic Action For all $A \in \Sigma$, $\mathcal{T}(A) = \{\langle A, \checkmark\rangle\}$

Basic Process $\mathcal{T}(YIELD) = \{\langle ?\rangle, \langle \checkmark\rangle\}$

Parallel Composition

$p\,\hat{}\langle\omega_1\rangle \parallel q\,\hat{}\langle\omega_2\rangle = \{r\,\hat{}\langle\omega_1 \& \omega_2\rangle \mid r \in (p \parallel\!\parallel q)\}$

$\mathcal{T}(P \parallel Q) = \{r \mid r \in (p \parallel q) \wedge p \in \mathcal{T}(P) \wedge q \in \mathcal{T}(Q)\}$

where

ω_1	✓	✓	✓	!	!	?
ω_2	✓	?	!	!	?	?
$\omega_1 \& \omega_2$	✓	?	!	!	!	?

Trace Semantics of Compensable Process

Terminating trace semantics

- The semantic model is a set of terminating trace pairs

Compensation Pair

$$p \div q = \begin{cases} (p, q) & p = p_1 \,\hat{}\, \langle \checkmark \rangle \\ (p, \checkmark) & p = p_1 \,\hat{}\, \langle \omega \rangle \wedge \omega \neq \checkmark \end{cases}$$

$$\mathcal{T}_c(P \div Q) = \{p \div q \mid p \in \mathcal{T}(P) \wedge q \in \mathcal{T}(Q)\} \cup \{(\langle ? \rangle, \langle \checkmark \rangle)\}$$

Sequential Composition

$$(p, p') \;;\; (q, q') = \begin{cases} (p_1 \,\hat{}\, q, q' \;;\; p') & p = p_1 \,\hat{}\, \langle \checkmark \rangle \\ (p, p') & p = p_1 \,\hat{}\, \langle \omega \rangle \wedge \omega \neq \checkmark \end{cases}$$

$$\mathcal{T}_c(PP \;;\; QQ) = \{(p, p') \;;\; (q, q') \mid (p, p') \in \mathcal{T}_c(PP) \wedge (q, q') \in \mathcal{T}_c(QQ)\}$$

Transaction Block

$$\mathcal{T}([PP]) = \{p \,\hat{}\, p' \mid (p \,\hat{}\, \langle ! \rangle, p') \in \mathcal{T}_c(PP)\} \cup \{p \,\hat{}\, \langle \checkmark \rangle \mid (p \,\hat{}\, \langle \checkmark \rangle, p') \in \mathcal{T}_c(PP)\}$$

Problems and Discussion (1)

The following laws do not hold:

$PP \ ; \ SKIPP = PP$

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

$[P \div P'] = P$ if P is non-yielding

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

$[P \div P' \ ; \ THROWW] = P \ ; \ P'$ if P 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 ($\sqcap$) and External ($\Box$) 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

$$
\begin{array}{lcl}
P & ::= & A \\
 & | & P \ ; \ P \\
 & | & P \sqcap P \\
 & | & P \Box P \\
 & | & P \underset{X}{\parallel} P \\
 & | & SKIP \\
 & | & THROW \\
 & | & YIELD \\
 & | & STOP \\
 & | & P \setminus X \\
 & | & P[\![R]\!] \\
 & | & P \rhd P \\
 & | & [PP]
\end{array}
\qquad
\begin{array}{lcl}
PP & ::= & P \div P \\
 & | & PP \ ; \ PP \\
 & | & PP \sqcap PP \\
 & | & PP \Box PP \\
 & | & PP \underset{X}{\parallel} PP \\
 & | & SKIPP \\
 & | & THROWW \\
 & | & YIELDD \\
 & | & PP \setminus X \\
 & | & PP[\![R]\!]
\end{array}
$$

Basic Idea of Stable Failures Semantics

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 $\checkmark$
  - 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: $\mathcal{T}_{\mathcal{S}}(P) : \mathcal{P} \to \mathbb{P}(\Sigma^{*\Omega})$
  - Failure set function: $\mathcal{F}_{\mathcal{S}}(P) : \mathcal{P} \to \mathbb{P}(\Sigma^{*\Omega} \times \mathbb{P}(\Sigma^{\Omega}))$
- Axioms of the semantic domain:

T is non-empty and prefix closed (1)

$$(s, X) \in F \Rightarrow s \in T \tag{2}$$

$$(s, X) \in F \wedge Y \subseteq X \Rightarrow (s, Y) \in F \tag{3}$$

$$(s, X) \in F \wedge \forall a \in Y \bullet s \hat{} \langle a \rangle \notin T \Rightarrow (s, X \cup Y) \in F \tag{4}$$

$$s \hat{} \langle \omega \rangle \in T \Rightarrow (s, \Sigma^{\Omega} \setminus \{\omega\}) \in F, \ where\ \omega \in \Omega \tag{5}$$

$$s \hat{} \langle \omega \rangle \in T \Rightarrow (s \hat{} \langle \omega \rangle, X) \in F, \ where\ \omega \in \Omega \wedge X \subseteq \Sigma^{\Omega} \tag{6}$$

Semantics of Parallel Composition - (1)

Trace synchronization, where $s_1, t_1 \in \Sigma^*$ and $\omega, \omega_1, \omega_2 \in \Omega$

$$s_1 \underset{X}{\parallel} t_1 \hat{} \langle \omega \rangle = \{\} \qquad s_1 \hat{} \langle \omega_1 \rangle \underset{X}{\parallel} t_1 \hat{} \langle \omega_2 \rangle = \{u \hat{} \langle \omega_1 \& \omega_2 \rangle \mid u \in s_1 \underset{X}{\parallel} t_1\}$$

Semantic definition

$$\begin{aligned} \mathcal{T}_{\mathcal{S}}(P \underset{X}{\parallel} Q) &= \{u \mid \exists s \in \mathcal{T}_{\mathcal{S}}(P), t \in \mathcal{T}_{\mathcal{S}}(Q) \bullet u \in (s \underset{X}{\parallel} t)\} \\ \mathcal{F}_{\mathcal{S}}(P \underset{X}{\parallel} Q) &= \{(u, E) \mid (u, E) \in (s, Y) \oplus (t, Z) \wedge \\ &\qquad \exists s, t \bullet (s, Y) \in \mathcal{F}_{\mathcal{S}}(P) \wedge (t, Z) \in \mathcal{F}_{\mathcal{S}}(Q)\} \end{aligned}$$

Laws need to hold:

$$THROW \underset{X}{\parallel} SKIP = YIELD \qquad THROW \underset{X}{\parallel} YIELD = THROW$$

$$YIELD \underset{X}{\parallel} SKIP = THROW$$

Semantics of Parallel Composition - (2)

Consider the parallel execution of P and Q

- $P \underset{X}{\parallel} Q$ can refuse an event in $X \cup \Omega$ if either P or Q can
- $P \underset{X}{\parallel} Q$ 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

- $P \underset{X}{\parallel} Q$ cannot terminate if either P or Q cannot terminate
- $P \underset{X}{\parallel} Q$ 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

Semantics of Parallel Composition - (3)

First example, $\Sigma = \{A, B\}$ and $A \underset{\{\}}{\parallel} (B \ ; \ THROW)$

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

Semantics of Parallel Composition - (3)

First example, $\Sigma = \{A, B\}$ and $A \underset{\{\}}{\parallel} (B \ ; \ THROW)$

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

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

- A has the failure $(\langle A\rangle, \{A, !, ?\})$
- $A \ ; \ THROW$ has the failure $(\langle A\rangle, \{A, \checkmark, ?\})$
- Both processes can terminate after executing $\langle A\rangle$, and the synchronized terminal event is !
- The synchronized failure set is $\{(\langle A\rangle, \{A, \checkmark, ?\})\}$

Semantics of Parallel Composition - (4)

Failure synchronization, where $(s, Y) \in \mathcal{F}_{\mathcal{S}}(P)$ and $(t, Z) \in \mathcal{F}_{\mathcal{S}}(Q)$

$$(s, Y) \oplus (t, Z) = \begin{cases} \{(u, Y \cup Z) \mid Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \wedge u \in s \underset{X}{\parallel} t\} \\ \qquad \textbf{if } (s, Y \cup \Omega) \in \mathcal{F}_{\mathcal{S}}(P) \vee (t, Z \cup \Omega) \in \mathcal{F}_{\mathcal{S}}(Q) \\ \{(u, (Y \cup Z) \setminus \Theta) \mid Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \wedge \\ \qquad u \in s \underset{X}{\parallel} t \wedge \Theta = rf(\omega_1, \omega_2)\} \\ \textbf{otherwise} \end{cases}$$

ω_1 is the terminal event P can perform to terminate

$$\forall (s, Y_1) \in \mathcal{F}_{\mathcal{S}}(P) \bullet Y \subseteq Y_1 \Rightarrow (\omega_1 \in \Omega \wedge \omega_1 \notin Y_1)$$

ω_2 is the terminal event Q can perform to terminate

$$\forall (t, Z_1) \in \mathcal{F}_{\mathcal{S}}(Q) \bullet Z \subseteq Z_1 \Rightarrow (\omega_2 \in \Omega \wedge \omega_2 \notin Z_1)$$

rf is the function for synchronizing ω_1 and ω_2

Semantics of Parallel Composition - (5)

In some cases, ω_1 or ω_2 may not exist, such as $(\langle\rangle, \{?\})$ of the process $SKIP \sqcap THROW$.

If ω_1 or ω_2 does not exist, we use ϵ to represent it.

$$rf(\omega_1, \omega_2) = \begin{cases} \{\omega_1 \& \omega_2\} & \omega_1 \in \Omega \wedge \omega_2 \in \Omega \\ \{\omega_1\} & \omega_1 \in \Omega \wedge \omega_2 = \epsilon \\ \{\omega_2\} & \omega_1 = \epsilon \wedge \omega_2 \in \Omega \\ \{\} & \omega_1 = \epsilon \wedge \omega_2 = \epsilon \end{cases}$$

More laws for parallel composition

$$\begin{array}{rcl} THROW \parallel P & = & P \,;\, THROW \\ THROW \parallel (YIELD \,;\, P) & = & THROW \sqcap (P \,;\, THROW) \end{array}$$

Outline

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

Semantic Model of Compensable Process

- 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)

Semantic Functions

- The semantics of PP can be calculated by the sematnics functions as $(\mathcal{T}^c(PP), \mathcal{F}^c(PP), \mathcal{C}(PP))$
  - The forward trace set function $\mathcal{T}^c(PP) : \mathcal{PP} \to \mathbb{P}(\Sigma^{*\Omega})$,
  - The forward failure set function $\mathcal{F}^c(PP) : \mathcal{PP} \to \mathbb{P}(\Sigma^{*\Omega} \times \mathbb{P}(\Sigma^{\Omega}))$
  - The compensation behavior set function $\mathcal{C}(PP) : \mathcal{PP} \to \mathbb{P}(\Sigma^*_{\Omega} \times \mathbb{P}(\Sigma^{*\Omega}) \times \mathbb{P}(\Sigma^{*\Omega} \times \mathbb{P}(\Sigma^{\Omega})))$
- 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.

Semantics of Compensation Pair

- 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

$$
\begin{aligned}
\mathcal{T}^c(P \div Q) &= \mathcal{T}_{\mathcal{S}}(P) \\
\mathcal{F}^c(P \div Q) &= \mathcal{F}_{\mathcal{S}}(P) \\
\mathcal{C}(P \div Q) &= \{(s, F^c, D^c) \mid \exists s \in (\mathcal{T}_{\mathcal{S}}(P) \cap \Sigma^*_\Omega) \bullet \\
&\qquad (s = t \,\hat{}\, \langle \checkmark \rangle \wedge T^c = \mathcal{T}_{\mathcal{S}}(Q) \wedge F^c = \mathcal{F}_{\mathcal{S}}(Q)) \vee \\
&\qquad (s \in \Sigma^*_{\{!,?\}} \wedge T^c = \mathcal{T}_{\mathcal{S}}(SKIP) \wedge F^c = \mathcal{F}_{\mathcal{S}}(SKIP))\}
\end{aligned}
$$

Semantics of Transaction Block

If an exception occurs in the forward behavior of PP, the compensation behavior will executed.

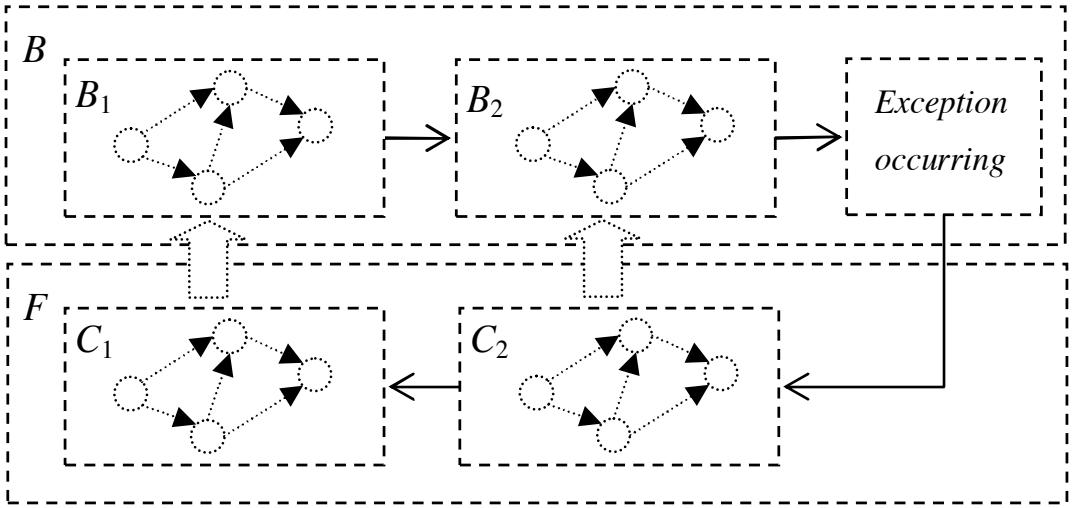
$$\begin{aligned}\mathcal{T}_{\mathcal{S}}([PP]) = \quad & (\mathcal{T}^c(PP) \cap \Sigma^{*\{\checkmark,?\}}) \cup \\ & \{s_1 \mid \exists (s, T^c, F^c) \in \mathcal{C}(PP) \bullet s = t \hat{} \langle ! \rangle \wedge s_2 \in T^c \wedge s_1 = t \hat{} s_2\}\end{aligned}$$

$$\begin{aligned}\mathcal{F}_{\mathcal{S}}([PP]) = \quad & \{(s, X) \mid s \in \Sigma^* \wedge (s, X \cup \{!\}) \in \mathcal{F}^c(PP)\} \cup \\ & \{(s_1, X_1) \mid \exists (s, T^c, F^c) \in \mathcal{C}(PP) \bullet \\ & \qquad (s \in \Sigma^*_{\{\checkmark,?\}} \wedge s_1 = s \wedge X_1 \subseteq \Sigma^\Omega) \vee \\ & \qquad (s = t \hat{} \langle ! \rangle \wedge (s_2, X_2) \in F^c \wedge s_1 = t \hat{} s_2 \wedge X_1 = X_2)\}\end{aligned}$$

Law

$$[P \div Q] \quad = \quad P \rhd SKIP$$

Semantics of Sequential Composition



The forward parts will be composed sequentially

$$\mathcal{T}_{\mathcal{S}}(PP\ ;\ QQ) = \mathcal{T}_{\mathcal{S}}(PP_f\ ;\ QQ_f) \quad \mathcal{F}^c(PP\ ;\ QQ) = \mathcal{F}_{\mathcal{S}}(PP_f\ ;\ QQ_f)$$

The compensation parts will be composed in the reversed order

$$\mathcal{C}(PP\ ;\ QQ) = \{(s, T^c, F^c) \mid \exists (s_1, PP_c) \in \mathcal{C}(PP), (s_2, QQ_c) \in \mathcal{C}(QQ) \bullet$$
$$(s_1 = t\,\hat{}\,\langle\checkmark\rangle \wedge s = t\,\hat{}\,s_2 \wedge T^c = \mathcal{T}_{\mathcal{S}}(QQ_c\ ;\ PP_c) \wedge F^c = \mathcal{F}_{\mathcal{S}}(QQ_c\ ;\ PP_c)) \vee$$
$$(s_1 \neq t\,\hat{}\,\langle\checkmark\rangle \wedge s = s_1 \wedge T^c = \mathcal{T}_{\mathcal{S}}(PP_c) \wedge F^c = \mathcal{F}_{\mathcal{S}}(PP_c))\}$$

Some Laws of Compensable Process

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

$[(P_1 \div Q_1 \underset{X}{\parallel} P_2 \div Q_2); THROWW] = (P_1 \underset{X}{\parallel} P_2) \ ; \ (Q_1 \underset{X}{\parallel} Q_2)$

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

$[(YIELDD \ ; \ P_1 \div Q_1 \ ; \ YIELDD \ ; \ P_2 \div Q_2) \parallel THROWW] =$
$SKIP \sqcap (P_1 \ ; Q_1) \sqcap (P_1 \ ; \ P_2 \ ; \ Q_2 \ ; \ Q_1)$

$[(YIELDD \ ; \ P_1 \div Q_1) \parallel (YIELDD \ ; \ P_2 \div Q_2) \parallel THROWW] =$
$SKIP \sqcap (P_1 \ ; \ Q_1) \sqcap (P_2 \ ; \ Q_2) \sqcap (P_1 \parallel P_2) \ ; \ (Q_1 \parallel Q_2)$

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

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?