Universal Temporal Concurrent Constraint Programming.

Carlos Olarte’s Ph.D Defense.


29 Sep 2009.
Motivation

Concurrent Systems are everywhere:

- **Engineering**: Security protocols, service oriented computing, mobile computing, synchronous systems.
- **Science**: Biological and chemical systems.
- **Arts**: Multimedia Interaction.
Motivation

Concurrent Systems are everywhere:

- **Engineering**: Security protocols, service oriented computing, mobile computing, synchronous systems.
- **Science**: Biological and chemical systems.
- **Arts**: Multimedia Interaction.

Models of Concurrency

Formal Models to describe and analyze concurrent systems. They must be:

- Simple.
- Expressive.
- Formal.
- Provide reasoning techniques.
Motivation

Concurrent Systems are everywhere:

- **Engineering**: Security protocols, service oriented computing, mobile computing, synchronous systems.
- **Science**: Biological and chemical systems.
- **Arts**: Multimedia Interaction.

Models of Concurrency

Formal Models to describe and analyze concurrent systems. They must be:

- Simple.
- Expressive.
- Formal.
- Provide reasoning techniques.

Some Examples: CCS [Mil89], the $\pi$-calculus [MPW92], CSP [Hoa85], ACP [BK85], **CCP** [Sar93].
Motivation

- Concurrent Constraint Programming (CCP) [Sar93] is a declarative model for Concurrency where agents interact by telling and asking information represented as constraints in a global store.
- The type of constraints and the entailment relation is given by a Constraint System (e.g. $x > 42 \models x > 0$).
Motivation

- Concurrent Constraint Programming (CCP) [Sar93] is a declarative model for Concurrency where agents interact by **telling** and **asking** information represented as **constraints** in a **global store**.

- The type of constraints and the **entailment** relation is given by a **Constraint System** (e.g. \( x > 42 \models x > 0 \)).
Concurrent Constraint Programming (CCP) [Sar93] is a declarative model for Concurrency where agents interact by telling and asking information represented as constraints in a global store.

The type of constraints and the entailment relation is given by a Constraint System (e.g. $x > 42 \models x > 0$).
Motivation

- Concurrent Constraint Programming (CCP) [Sar93] is a declarative model for Concurrency where agents interact by telling and asking information represented as constraints in a global store.
- The type of constraints and the entailment relation is given by a Constraint System (e.g. \( x > 42 \models x > 0 \)).

```
42 <temperature<70
```

```
ask temperature = 50 then P
```
Our Goal

- We aim at developing a theory for a CCP-based model for the specification of mobile reactive systems where logic and behavioral approaches coexist coherently.
Our Goal

- We aim at developing a theory for a CCP-based model for the specification of mobile reactive systems where logic and behavioral approaches coexist coherently.

Criteria:
- **Declarative**: allowing for reachability analysis using deduction in logic.
- **Determinism**: which is the source of CCP’s elegant and simple semantic characterizations.
- **Applications** in emergent application areas.
Local variables define boundaries of interaction. How can we change the communication structure of the processes?

- Variables as communication channels [Sar93].
  - channels can be used only once.
- Atomic CCP with pattern matching [LM92].
  - Atomic CCP is non-deterministic.
- Adding linear parametric asks (LCC [FRS01, SL92]).
  - Non-determinism is introduced.
- Adding persistent parametric asks.
  - Not all the inputs must be persistent.

Our approach: A CCP-based calculus with temporary parametric asks: The Universal Timed CCP calculus (utcc).
Our Contributions

Reasoning Techniques for utcc:

- A novel **symbolic semantics** based on temporal formulae.
- Interpretation of utcc processes as formulae in Pnueli’s FLTL.
- A denotational semantics based on **closure operators**.
- Abstract semantics as basis for the static analysis of utcc programs.
Our Contributions

Reasoning Techniques for utcc:

- A novel **symbolic semantics** based on temporal formulae.
- Interpretation of utcc processes as formulae in Pnueli’s **FLTL**.
- A denotational semantics based on **closure operators**.
- **Abstract semantics** as basis for the static analysis of utcc programs.

Theoretical Results:

- We prove the **undecidability** of the Monadic fragment of Pnueli’s First-Order LTL.
Our Contributions

Reasoning Techniques for utcc:

- A novel *symbolic semantics* based on temporal formulae.
- Interpretation of utcc processes as formulae in Pnueli’s FLTL.
- A denotational semantics based on *closure operators*.
- *Abstract semantics* as basis for the static analysis of utcc programs.

Theoretical Results:
- We prove the *undecidability* of the Monadic fragment of Pnueli’s First-Order LTL.

Applications:
- Closure operator semantics for languages for *security*.
- Declarative interpretation and temporal extensions for *sessions*.
- Modeling of *multimedia* interactive systems.
Outline

1. Intuitive Description and SOS
2. Symbolic Semantics
3. Undecidability of FLTL
4. Denotational Semantics for $\text{utcc}$
5. Applications
6. Abstract Semantics and Static Analysis of $\text{utcc}$
7. Concluding Remarks
Outline

1. Intuitive Description and SOS
2. Symbolic Semantics
3. Undecidability of FLTL
4. Denotational Semantics for utcc
5. Applications
6. Abstract Semantics and Static Analysis of utcc
7. Concluding Remarks
The tcc Model [SJG94]

1. Receives a stimulus (i.e. a constraint) from the environment.
The tcc Model [SJG94]

1. Receives a stimulus (i.e a constraint) from the environment.
2. Computes a CCP process in the current time-unit and wait for stability.

Note: Stores are not automatically transferred from a time unit to the next one.
The tcc Model [SJG94]

1. Receives a **stimulus** (i.e. a constraint) from the environment.
2. Computes a CCP process in the current **time-unit** and wait for stability.
3. ** Responds ** with the resulting store.
The tcc Model [SJG94]

1. Receives a **stimulus** (i.e. a constraint) from the environment.
2. Computes a CCP process in the current *time-unit* and wait for stability.
3. **Responds** with the resulting store.
4. Executes the **Residual** process in the *next* time-unit.

* **Note:** Stores are not automatically transferred from a time unit to the next one.
The tcc calculus [SJG94]
Syntax

\[ P, Q ::= \text{skip} \mid \text{tell}(c) \mid \text{when } c \text{ do } P \mid P \parallel Q \mid (\text{local } \vec{x}; c) P \mid \text{next } P \mid \text{unless } c \text{ next } P \mid ! P \]

- **tell**\((c)\): adds \(c\) to the store in the current time interval.
- **when** \(c\) **do** \(P\): executes \(P\) if \(c\) can be deduced from the current store.
- \((\text{local } \vec{x}; c) P\): behaves like \(P\) but the information about variables in \(\vec{x}\) is local to \(P\)
- **next** \(P\): executes \(P\) in the next time unit.
- **unless** \(c\) **next** \(P\): executes \(P\) in the next time unit if \(c\) cannot be entailed now.
- **!** \(P\): Unboundedly many copies of \(P\), one at a time.
Abstractions and the utcc Calculus

Example

Let $Q = \text{tell}(\text{out}(42))$ and $P = \text{when} \ \text{out}(x) \ \text{do} \ R$.

- $x$ is not a place holder in $P$
- $\text{out}(42) \not\models \text{out}(x)$. Then $P \parallel Q \not\rightarrow$. 
Abstractions and the utcc Calculus

Example
Let $Q = \text{tell}(\text{out}(42))$ and $P = \text{when } \text{out}(x) \text{ do } R$.
- $x$ is not a place holder in $P$
- $\text{out}(42) \not\equiv \text{out}(x)$. Then $P \parallel Q \not\rightarrow$.

The idea of Abstractions in utcc

The abstraction construct $S = (\text{abs } \vec{x}; c) R$:
- $S$ can be seen as a $\lambda$-abstraction of $R$ on $\vec{x}$ with guard $c$.
- $S$ is a parametric ask that executes $R[\vec{x}/\vec{t}]$ for each $\vec{t}$ s.t. $c[\vec{x}/\vec{t}]$ can be deduced from the current store. E.g., $S \parallel Q \rightarrow R[42/x]$.
- The variables in $\vec{x}$ can be seen as the formal parameters of $R$.
- Logical point of view: $S$ corresponds to a formula $\forall \vec{x}.(c \Rightarrow F)$. 
Operational Semantics

Internal transitions (\(\rightarrow\))

\[\text{**R}_{\text{TELL}}\]
\[\langle \text{tell}(c), d \rangle \rightarrow \langle \text{skip}, d \land c \rangle\]

\[\text{**R}_{\text{PAR}}\]
\[\langle P, c \rangle \rightarrow \langle P', d \rangle\]
\[\langle P \parallel Q, c \rangle \rightarrow \langle P' \parallel Q, d \rangle\]

\[\text{**R}_{\text{ABS}}\]
\[d \models_\Delta c[\vec{t}/\vec{x}]\quad [\vec{t}/\vec{x}] \text{ is admissible.}\]
\[\langle (\text{abs } \vec{x}; c) P, d \rangle \rightarrow \langle P[\vec{t}/\vec{x}] \parallel (\text{abs } \vec{x}; c \land \vec{x} \not= \vec{t}) P, d \rangle\]

Observable transitions (\(\Rightarrow\))

\[\text{**R}_{\text{OBS}}\]
\[\langle P, c \rangle \rightarrow^* \langle Q, d \rangle \notightarrow\]
\[P \xrightarrow{(c,d)} F(Q)\]
\[
F((\text{abs } \vec{x}; c) Q) = \text{skip}
\]
\[
F(\text{next } Q) = Q
\]
\[
F(\text{unless } c \text{ next } Q) = Q
\]
Input-output Behavior

- $P \xrightarrow{(c,c')} Q$ : $P$ under input $c$ outputs $c'$ and executes $Q$ in the next time-unit.

- Similarly, $P \xrightarrow{(\alpha,\alpha')}$, or $(\alpha,\alpha') \in io(P)$, whenever $\alpha = c_1.c_2....$, $\alpha' = c'_1.c'_2....$ and $P = P_1 \xrightarrow{(c_1,c'_1)} P_2 \xrightarrow{(c_2,c'_2)} ...P_i \xrightarrow{(c_i,c'_i)} ...$. 

Theorem (Determinism) 

Let $\alpha,\beta$ and $\beta'$ be sequences of constraints. If both $(\alpha,\beta), (\alpha,\beta') \in io(P)$ then for all $i > 0$, $\beta(i) \equiv \beta'(i)$. 

Intuitive Description and SOS

Carlos Olarte's PhD Defense, LIX, École Polytechnique.
Input-output Behavior

- \( P \xrightarrow{(c,c')} Q \): \( P \) under input \( c \) outputs \( c' \) and executes \( Q \) in the next time-unit.
- Similarly, \( P \xrightarrow{(\alpha,\alpha')} \), or \((\alpha,\alpha') \in io(P)\), whenever \( \alpha = c_1.c_2... \), \( \alpha' = c_1'.c_2'... \), and \( P = P_1 \xrightarrow{(c_1,c_1')} P_2 \xrightarrow{(c_2,c_2')} ...P_i \xrightarrow{(c_i,c_i')} ... \).

Theorem (Determinism)

Let \( \alpha, \beta \) and \( \beta' \) be sequences of constraints. If both \((\alpha, \beta), (\alpha, \beta') \in io(P)\) then for all \( i > 0 \), \( \beta(i) \equiv \beta'(i) \).
Some Examples
Communication using global channels

$\begin{align*}
    Alice & \xrightarrow{\{m\}_B} Bob \\
    A & = (\text{local } m)(\text{tell}(\text{out}(\{m\}_B))) \\
    B & = (\text{abs } x; \text{out}(:, x}_B)) B'
\end{align*}$
Some Examples
Communication using global channels

\[
\begin{align*}
A & = (\text{local } m)(\text{tell}(\text{out}(\{m\}_B))) \\
B & = (\text{abs } x; \text{out}(\{x\}_B)) B'
\end{align*}
\]

<table>
<thead>
<tr>
<th>Process</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \parallel B )</td>
<td>true</td>
</tr>
</tbody>
</table>
Some Examples
Communication using global channels

\[ Alice \xrightarrow{\{m\}_B} Bob \]

\[ A = (\text{local } m)(\text{tell}(\text{out}(\{m\}_B))) \]
\[ B = (\text{abs } x; \text{out}(\{x\}_B)) B' \]

<table>
<thead>
<tr>
<th>Process</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \parallel B$</td>
<td>true</td>
</tr>
<tr>
<td>$(\text{local } m)(\text{tell}(\text{out}({m}_B)) \parallel B)$</td>
<td>true</td>
</tr>
</tbody>
</table>
Some Examples
Communication using global channels

\[\text{Alice} \xrightarrow{{m}_B} \text{Bob}\]

\[A = (\text{local } m)(\text{tell}(\text{out}({m}_B)))\]
\[B = (\text{abs } x; \text{out}({x}_B)) B'\]

<table>
<thead>
<tr>
<th>Process</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A \parallel B)</td>
<td>true</td>
</tr>
<tr>
<td>((\text{local } m)(\text{tell}(\text{out}({m}_B))) \parallel B)</td>
<td>true</td>
</tr>
<tr>
<td>((\text{local } m; \text{out}({m}_B))(B))</td>
<td>(\exists m.\text{out}({m}_B))</td>
</tr>
</tbody>
</table>
Some Examples

Communication using global channels

\[
\text{Alice} \xrightarrow{\{m\}_B} \text{Bob}
\]

\[
A = (\text{local } m)(\text{tell}(\text{out}(\{m\}_B)))
\]

\[
B = (\text{abs } x; \text{out}(\{x\}_B)) B'
\]

<table>
<thead>
<tr>
<th>Process</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \parallel B$</td>
<td>true</td>
</tr>
<tr>
<td>$(\text{local } m)(\text{tell}(\text{out}({m}_B)) \parallel B)$</td>
<td>true</td>
</tr>
<tr>
<td>$(\text{local } m; \text{out}({m}_B))(B)$</td>
<td>$\exists m.(\text{out}({m}_B))$</td>
</tr>
<tr>
<td>$(\text{local } m; \text{out}({m}_B))(B'[m/x] \parallel B'')$</td>
<td>$\exists m.(\text{out}({m}_B))$</td>
</tr>
</tbody>
</table>

where $B'' = (\text{abs } x; \text{out}(\{x\}_B \land x \neq m)) B'$.
Some Examples
Communication of Local Names

\[
P_\pi = (\nu b)(\bar{a}b.b(y).R_\pi)
Q_\pi = a(x). (\nu c)(\bar{x}c)
\]

\[
P_\pi \mid Q_\pi \rightarrow^* (\nu b, c)R_\pi[c/y]
\]
Some Examples
Communication of Local Names

\[ P_\pi = (\nu b)(\bar{a}b.b(y).R_\pi) \]
\[ Q_\pi = a(x).(\nu c)(\bar{x}c) \]

\[ P_\pi \parallel Q_\pi \rightarrow^*_\pi (\nu b, c)R_\pi[c/y] \]

\[ P = (\textbf{local } b)(\textbf{tell}\ out(a, b)) \parallel (\textbf{abs } y; \textbf{out}(b, y)) R) \]
\[ Q = (\textbf{abs } x; \textbf{out}(a, x)) (\textbf{local } c)(\textbf{tell}\ out(x, c))) \]
Some Examples
Communication of Local Names

\[
P^\pi = (\nu b)(\overline{a}b.b(y).R^\pi) \\
Q^\pi = a(x).((\nu c)(\overline{x}c) \\

P^\pi \mid Q^\pi \rightarrow^* (\nu b, c)R^\pi[c/y] \\

P = (\text{local } b)(\text{tell(out}(a, b)) \parallel (\text{abs } y; \text{out}(b, y)) R) \\
Q = (\text{abs } x; \text{out}(a, x))(\text{local } c)(\text{tell(out}(x, c)))

P \parallel Q \\
\rightarrow (\text{local } b; \text{out}(a, b))((\text{abs } y; \text{out}(b, y)) R \parallel \\
(\text{abs } x; \text{out}(a, x)) Q') \\
\rightarrow (\text{local } b; \text{out}(a, b))((\text{abs } y; \text{out}(b, y)) R \parallel \\
(\text{local } c)(\text{tell(out}(b, c)) \\
\rightarrow (\text{local } b, c; \text{out}(a, b) \land \text{out}(b, c))((\text{abs } y; \text{out}(b, y)) R) \\
\rightarrow (\text{local } b, c; \text{out}(a, b) \land \text{out}(b, c))(R[c/y])
\]
Some Examples
Encoding Recursive Definitions

The variables \( \vec{x} \) in \((\text{abs } \vec{x}; c) \ P\) can be seen as the \textit{formal parameters} of \( P \).

- Proc. definitions: \( \neg p(\vec{y}) \overset{\text{def}}{=} P \vdash =! (\text{abs } \vec{y}; \text{call}_p(\vec{y})) \hat{P} \)
  where \( \hat{P} \) is obtained by replacing \( p(\vec{x}) \) with \textbf{tell}(\text{call}_p(\vec{x})) \).
Some Examples

Encoding Recursive Definitions

The variables $\vec{x}$ in $(\text{abs } \vec{x}; c) P$ can be seen as the formal parameters of $P$.

- Proc. definitions: $p(\vec{y}) \overset{\text{def}}{=} P\neg \Rightarrow ! (\text{abs } \vec{y}; \text{call}_p(\vec{y})) \hat{P}$
  where $\hat{P}$ is obtained by replacing $p(\vec{x})$ with $\text{tell}(\text{call}_p(\vec{x}))$.

A simple Example

- Definition:

  $! (\text{abs } N, M, X; \text{fact}(N, M, X))$ when $N \leq 1$ do tell($X = M$) \parallel
  when $N > 1$ do tell($\text{fact}(N - 1, N \times M, X)$)

- Call:

  $\langle \text{tell}(\text{fact}(3, 1, X)), \text{true} \rangle \longrightarrow^* \langle P, X = 6 \rangle$
Infinite Behavior
No observable transition!!!

The `abs` construct may introduce infinitely many internal reductions:

- **Loops**: \((\text{abs } x; c(x)) \text{ tell}(c(x + 1))\)
- **Infinitely many substitutions**: \(P = (\text{abs } x; x > 1) \ R.\)
Infinite Behavior

No observable transition!!!

The \texttt{abs} construct may introduce infinitely many internal reductions:

- **Loops**: \((\texttt{abs } x; c(x)) \texttt{tell}(c(x + 1))\)
- **Infinitely many substitutions**: \(P = (\texttt{abs } x; x > 1) R\).

**Example (Message Composition)**

\[
P = (\texttt{abs } x, y; \texttt{out}(x) \land \texttt{out}(y)) \texttt{tell}(\texttt{out}(\{x, y\}))
\]

What do we observe from \(P \parallel \texttt{tell}(\texttt{out}(m))\) ? :

\[
\text{out}(\{m, m\}), \text{out}(\{m, \{m, m\}\}), \text{out}(\{m, \{m, \{m, m\}\}\})\ldots
\]
Outline

1 Intuitive Description and SOS
2 Symbolic Semantics
3 Undecidability of FLTL
4 Denotational Semantics for utcc
5 Applications
6 Abstract Semantics and Static Analysis of utcc
7 Concluding Remarks
Symbolic Semantics for utcc

\[ P = (\text{abs } x, y; \text{out}(x) \land \text{out}(y)) \text{tell}(\text{out}({x, y})) \]

Symbolically

\[ \langle P, \text{out}(m) \rangle \xrightarrow{s} \langle \text{skip}, \text{out}(m) \land \forall x, y : (\text{out}(x) \land \text{out}(y) \Rightarrow \text{out}({x, y})) \rangle \]
Symbolic Semantics for \texttt{utcc}

\[ P = (\texttt{abs } x, y; \texttt{out}(x) \land \texttt{out}(y)) \texttt{tell}(\texttt{out} \{x, y\}) \]

Symbolically
\[ \langle P, \texttt{out}(m) \rangle \rightarrow_s \langle \texttt{skip}, \texttt{out}(m) \land \forall x, y : (\texttt{out}(x) \land \texttt{out}(y) \Rightarrow \texttt{out} \{x, y\}) \rangle \]

Temporal Dependencies

\[ P = (\texttt{abs } x, y; \texttt{out}(x) \land \texttt{out}(y)) \texttt{next tell}(\texttt{out} \{x, y\}) \]

\[ \langle P, \texttt{out}(m) \rangle \not\rightarrow_s \Rightarrow_s \langle \texttt{skip}, \ominus \texttt{out}(m) \land \forall x, y : (\ominus (\texttt{out}(x) \land \texttt{out}(y)) \Rightarrow \texttt{out} \{x, y\}) \rangle \]
Symbolic Semantics

Symbolic Reductions

\[
\begin{align*}
R_{\text{ABS-SYM}} &: \quad \langle P, \exists \bar{x}d \rangle \rightarrow_s \langle P', \exists \bar{x}d \land d' \rangle \\
& \quad \langle (\text{abs } \bar{x}; c)\ P, d \rangle \rightarrow_s \langle (\text{abs } \bar{x}; c)\ P', d \land \forall \bar{x}(c \Rightarrow d') \rangle \\
R_{\text{OBS-SYM}} &: \quad \langle P, c \rangle \rightarrow_s^* \langle Q, d \rangle \not\rightarrow_s \\
& \quad P \xrightarrow{(c,d)} F_s(Q, d)
\end{align*}
\]

Symbolic Future Function : \( F_s(P, d) = \text{tell}(\ominus d) \parallel F'_s(P) \)

\[
F'_s(P) = \begin{cases} 
(\text{abs } \bar{x}; \ominus c)\ F_s(P) & \text{if } P = (\text{abs } \bar{x}; c)\ P \\
(\text{local } \bar{x}; \ominus c)\ F_s(Q) & \text{if } P = (\text{local } \bar{x}; c)\ Q
\end{cases}
\]

Theorem (Semantic Correspondence)
Let \( P \) be an abstracted-unless free process. The symbolic and the operational outputs of \( P \) entail the same basic constraints.
Symbolic Semantics

Symbolic Reductions

\[ R_{\text{ABS-SYM}} \]

\[
\langle P, \exists \bar{x} d \rangle \rightarrow_s \langle P', \exists \bar{x} d \land d' \rangle
\]

\[
\langle (\text{abs } \bar{x}; c) P, d \rangle \rightarrow_s \langle (\text{abs } \bar{x}; c) P', d \land \forall \bar{x}(c \Rightarrow d') \rangle
\]

\[ R_{\text{OBS-SYM}} \]

\[
\langle P, c \rangle \rightarrow^* \langle Q, d \rangle \rightarrow_s F_s(Q, d)
\]

Symbolic Future Function: \( F_s(P, d) = \text{tell}(\ominus d) \parallel F'_s(P) \)

\[
F'_s(P) = \begin{cases} 
(\text{abs } \bar{x}; \ominus c) F_s(P) & \text{if } P = (\text{abs } \bar{x}; c) P \\
(\text{local } \bar{x}; \ominus c) F_s(Q) & \text{if } P = (\text{local } \bar{x}; c) Q 
\end{cases}
\]

Theorem (Semantic Correspondence)

Let \( P \) be an abstracted-unless free process. The symbolic and the operational outputs of \( P \) entail the same basic constraints.
Definition (FLTL Syntax)

\[ F, G, \ldots := c \mid F \land G \mid \neg F \mid \exists x F \mid \ominus F \mid \circ F \mid \Box F \]

\(c\) is a constraint in \(\mathcal{L}\). \(\Diamond F = \neg \Box \neg F\) (eventually \(F\)).
FLTL Correspondence (Declarative view of Processes)

Definition (FLTL Syntax)

\[ F, G, \ldots := c \mid F \land G \mid \neg F \mid \exists x F \mid \lozenge F \mid \circ F \mid \square F \]

\( c \) is a constraint in \( \mathcal{L} \). \( \lozenge F = \neg \square \neg F \) (eventually \( F \)).

Definition

FLTL Interpretation of utccProcesses

\[
\begin{align*}
\llbracket \text{skip} \rrbracket &= \text{true} \\
\llbracket (\text{abs } \bar{y}; c) P \rrbracket &= \forall \bar{y} (c \Rightarrow \llbracket P \rrbracket) \\
\llbracket (\text{local } \bar{x}; c) P \rrbracket &= \exists \bar{x} (c \land \llbracket P \rrbracket) \\
\llbracket \text{unless } c \text{ next } P \rrbracket &= c \lor \circ \llbracket P \rrbracket
\end{align*}
\]

\[
\begin{align*}
\llbracket \text{tell}(c) \rrbracket &= c \\
\llbracket P \parallel Q \rrbracket &= \llbracket P \rrbracket \land \llbracket Q \rrbracket \\
\llbracket \text{next } P \rrbracket &= \circ \llbracket P \rrbracket \\
\llbracket ! P \rrbracket &= \square \llbracket P \rrbracket
\end{align*}
\]

Theorem (Logic Correspondence)

If \( P \) is a monotonic process, \( \llbracket P \rrbracket \models T \) \( c \) iff \( P \downarrow c \).
FLTL Correspondence (Declarative view of Processes)

Definition (FLTL Syntax)

\[ F, G, \ldots := c \mid F \land G \mid \neg F \mid \exists x F \mid \lozenge F \mid \circ F \mid \Box F \]

\( c \) is a constraint in \( \mathcal{L} \). \( \lozenge F = \neg \Box \neg F \) (eventually \( F \)).

Definition

FLTL Interpretation of utcc Processes

\[
\begin{align*}
\llbracket \text{skip} \rrbracket & = \text{true} \\
\llbracket (\text{abs } \bar{y}; c) P \rrbracket & = \forall \bar{y} (c \Rightarrow \llbracket P \rrbracket) \\
\llbracket (\text{local } \bar{x}; c) P \rrbracket & = \exists \bar{x} (c \land \llbracket P \rrbracket) \\
\llbracket \text{unless } c \text{ next } P \rrbracket & = c \lor \circ \llbracket P \rrbracket
\end{align*}
\]

\[
\begin{align*}
\llbracket \text{tell}(c) \rrbracket & = c \\
\llbracket P \parallel Q \rrbracket & = \llbracket P \rrbracket \land \llbracket Q \rrbracket \\
\llbracket \text{next } P \rrbracket & = \circ \llbracket P \rrbracket \\
\llbracket \neg P \rrbracket & = \Box \llbracket P \rrbracket
\end{align*}
\]

Theorem (Logic Correspondence)

*If \( P \) is a monotonic process, \( \llbracket P \rrbracket \models T \lozenge c \) iff \( P \Downarrow_s^C \).*
Outline

1. Intuitive Description and SOS
2. Symbolic Semantics
3. Undecidability of FLTL
4. Denotational Semantics for utcc
5. Applications
6. Abstract Semantics and Static Analysis of utcc
7. Concluding Remarks
Undecidability of Monadic FLTL

**Theorem (utcc is Turing powerful)**

*The Minsky machine \( M(0,0) \) halts iff \([M(0,0)] \downarrow^{\text{halt}}\)*

Let \( F \) be the FLTL formulae corresponding to \([M(0,0)]\):

\[ F \text{ is a monadic FLTL formula without functions nor equality.} \]

Let \( G = (F = \Rightarrow \Box \text{running}) \).

\( G \) is valid iff \( M \) never halts.

**Theorem (Incompleteness of Pnueli's FLTL)**

Monadic FLTL without function symbols nor equality is incomplete.

Carlos Olarte's PhD Defense, LIX, École Polytechnique.
Undecidability of Monadic FLTL

Theorem (utcc is Turing powerful)

The Minsky machine $M(0, 0)$ halts iff $\llbracket M(0, 0) \rrbracket \downarrow^{\text{halt}}$

Monadic FLTL is Incomplete

Let $F$ be the FLTL formulae corresponding to $\llbracket M(0, 0) \rrbracket$:
- $F$ is a monadic FLTL formula without functions nor equality.
- Let $G = (F \implies \Box \text{running})$. $G$ is valid iff $M$ never halts.
Undecidability of Monadic FLTL

Theorem (utcc is Turing powerful)

*The Minsky machine* $M(0,0)$ *halts iff* $\llbracket M(0,0) \rrbracket \downarrow_{halt}$

Monadic FLTL is Incomplete

Let $F$ be the FLTL formulae corresponding to $\llbracket M(0,0) \rrbracket$:
- $F$ is a monadic FLTL formula without functions nor equality.
- Let $G = (F \implies \Box \text{running})$. *$G$ is valid iff $M$ never halts*.

Theorem (Incompleteness of Pnueli’s FLTL)

*Monadic FLTL without function symbols nor equality is incomplete*.
Our result in Context

1. [Mer92] proved the Monadic fragment of FLTL to be decidable!!.
   - Quantification of flexible variables was not allowed in [Mer92].

2. [Val05] conjectures that the negation-free restriction can be dropped and still obtain decidability fragments of FLTL.
   - With negation the FLTL in [Val05] is the same studied here.

Our result shows (1) and (2) to be necessary to obtain decidability.
Outline

1 Intuitive Description and SOS
2 Symbolic Semantics
3 Undecidability of FLTL
4 Denotational Semantics for utcc
5 Applications
6 Abstract Semantics and Static Analysis of utcc
7 Concluding Remarks
Strongest Postcondition
Symbolic Input-output Relation

- **Input-output Relation** : \((w, v)\) s.t. \(P \xrightarrow{(w,v)} s\)
- **Strongest Postcondition** : \(w \in sp_s(P)\) if \(P\) cannot add any information to \(w\).

If \(P\) is monotonic, \(io_s(P)\) is a closure operator, i.e., a function satisfying extensiveness, idempotence and monotonicity whose set of fixed points is \(sp_s(P)\):

\[(w, w') \in io_s(P) \iff w' = \min(sp_s(P) \cap \{w \mid s \leq w\})\]
Denotational Semantics

Compositional Characterization of the $sp_s(\cdot)$ relation.

\[ \text{D}_{\text{TELL}} \quad \llbracket \text{tell}(c) \rrbracket = \{ e.w \mid e \models_T c \} \]

\[ \text{D}_{\text{PAR}} \quad \llbracket P \parallel Q \rrbracket = \llbracket P \rrbracket \cap \llbracket Q \rrbracket \]

\[ \text{D}_{\text{LOC}} \quad \llbracket (\text{local} \ \vec{x}; c) P \rrbracket = \{ w \mid \text{there exists an } \vec{x}\text{-variant } w' \text{ of } w \text{ s.t. } w'(1) \models_T c \text{ and } w' \in \llbracket P \rrbracket \} \]

\[ \text{D}_{\text{ABS}} \quad \llbracket (\text{abs} \ \vec{x}; c) P \rrbracket = \{ w \mid \text{for every } \vec{x}\text{-variant } w' \text{ of } w \text{ if } w'(1) \models_T c \text{ and } w' \succeq (\vec{x} = \vec{t})^\omega \text{ for some admissible } \vec{t} \text{ then } w' \in \llbracket P \rrbracket \} \]
Denotational Semantics

Compositional Characterization of the $sp_s(\cdot)$ relation.

$D_{\text{TELL}} \quad [\text{tell}(c)] = \{e.w \mid e \models_T c\}$

$D_{\text{PAR}} \quad [P \parallel Q] = [P] \cap [Q]$

$D_{\text{LOC}} \quad [(\text{local } \vec{x} ; c) P] = \{w \mid \text{there exists an } \vec{x}\text{-variant } w' \text{ s.t. } w'(1) \models_T c \text{ and } w' \in [P]\}$

$D_{\text{ABS}} \quad [(\text{abs } \vec{x}; c) P] = \{w \mid \text{for every } \vec{x}\text{-variant } w' \text{ of } w \text{ if } w'(1) \models_T c \text{ and } w' \succeq (\vec{x} = \vec{t})\omega \text{ for some admissible } \vec{t} \text{ then } w' \in [P]\}$

Theorem (Full Abstraction)

Let $P, Q$ be monotonic processes. It holds:

- $P \approxio Q$ iff $[P] = [Q]$. 

Carlos Olarte’s PhD Defense, LIX, École Polytechnique.
Outline

1. Intuitive Description and SOS
2. Symbolic Semantics
3. Undecidability of FLTL
4. Denotational Semantics for \texttt{utcc}
5. Applications
6. Abstract Semantics and Static Analysis of \texttt{utcc}
7. Concluding Remarks
Closure operator semantics for Sec. Languages.

SCCP Syntax

Values  \( v, v' \) ::= \( n \mid x \)
Keys  \( k \) ::= \( pub(v) \mid priv(v) \)
Messages  \( M, N \) ::= \( v \mid k \mid X \mid \{M, N\} \mid \{M\}_k \)
Processes  \( R \) ::= \( \text{nil} \)
\( \mid \text{new}(x)R \)
\( \mid \text{out}(M).R \)
\( \mid \text{in } (\vec{x})[M].R \)
\( \mid !R \)
\( \mid R_i \parallel R_j \)
Closure operator semantics for Sec. Languages.

SCCP Syntax

Values \( v, v' \) ::= \( n | x \)

Keys \( k \) ::= \( pub(v) | priv(v) \)

Messages \( M, N \) ::= \( v | k | X | \{ M, N \} | \{ M \}_k \)

Processes \( R \) ::= \( \text{nil} \quad \implies \text{skip} \)

| \( \text{new}(x).R \) & \( \implies (\text{local } x)[R] \)
| \( \text{out}(M).R \) & \( \implies !\text{tell(out}(M)) \| \text{next}[R] \)
| \( \text{in}(\bar{x})[M].R \) & \( \implies (\text{abs } \bar{x}; \text{out}(M)) \text{next}[R] \)
| \( !R \) & \( \implies ![R] \)
| \( R_i \| R_j \) & \( \implies [R_i] \| [R_j] \)
Closure operator semantics for Sec. Languages.

**SCCP Syntax**

Values \( \nu, \nu' \) ::= \( n \mid x \)

Keys \( k \) ::= \( \text{pub}(\nu) \mid \text{priv}(\nu) \)

Messages \( M, N \) ::= \( \nu \mid k \mid X \mid \{M, N\} \mid \{M\}_k \)

Processes \( R \) ::= \( \text{nil} \implies \text{skip} \)

\[
\begin{align*}
\text{new}(x)R & \implies (\text{local } x)[[R]] \\
\text{out}(M).R & \implies !\text{tell}(\text{out}(M)) || \text{next}[[R]] \\
\text{in } (\vec{x})[M].R & \implies (\text{abs } \vec{x}; \text{out}(M)) \text{ next }[[R]] \\
!R & \implies ![R] \\
R_i || R_j & \implies [[R_i]] || [[R_j]]
\end{align*}
\]

**Security Constraint System**

<table>
<thead>
<tr>
<th>PRJ</th>
<th>( F \models \text{out}({m_1, m_2}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENC</td>
<td>( F \models \text{out}(m_{1/2}) )</td>
</tr>
<tr>
<td>ENC</td>
<td>( F \models \text{out}(m_1) ) ( F \models \text{out}(m_2) )</td>
</tr>
<tr>
<td>ENC</td>
<td>( F \models \text{out}({m}_{m_2}) )</td>
</tr>
</tbody>
</table>

| DEC | \( F \models \text{out}(k^{-1}) \) \( F \models \text{out}(\{m\}_k) \) |
| DEC | \( F \models \text{out}(m) \) |
| PAIR | \( F \models \text{out}(m_1) \) \( F \models \text{out}(m_2) \) |
| PAIR | \( F \models \text{out}(\{m_1, m_2\}) \) |
An Example

Denning-Sacco key distribution protocol:

\[
\begin{align*}
msg_1 & \quad A \rightarrow B : \ (A, m)_{pub(B)} \\
msg_2 & \quad B \rightarrow A : \ n_{pub(m)}
\end{align*}
\]

\[
\begin{align*}
Init(A, B) &= \text{! new}(m)\text{out}((m, A)_{pub(B)})\cdot\text{nil} \\
Resp(B) &= \text{! in } (x, u)[((x, u)_{priv(B)})\cdot\text{new}(n)(\text{out}(n)_{pub(u)})\cdot\text{nil}) \parallel \text{! in } [n]\cdot\text{out(attack)}\cdot\text{nil}
\end{align*}
\]
An Example

Denning-Sacco key distribution protocol:

\[ \text{msg}_1 \quad A \to B : \quad \{(A, m)\}_{\text{pub}(B)} \]
\[ \text{msg}_2 \quad B \to A : \quad \{n\}_{\text{pub}(m)} \]

\[ \text{Init}(A, B) = \text{! new}(m)\text{out}([\{(m, A)\}_{\text{pub}(B)}] \cdot \text{nil} \]
\[ \text{Resp}(B) = \text{! in } (x, u)\lbrack\{(x, u)\}_{\text{priv}(B)}\rbrack \cdot \]
\[ \text{new}(n)\text{out}([n]_{\text{pub}(u)} \cdot \text{nil}) \parallel \text{! in } [n]\text{.out}(\text{attack})\cdot \text{nil} \]

Proposition

Let \( R \) be a SCCP process, \( P \) the utcc process representing \( R \) and
\( f = \lbrack P \rbrack \cap \lbrack \text{! when out}(\text{attack}) \text{ do ! tell}(\text{false})\rbrack \).

- **Symbolic Output**: \( P \downarrow^{\text{attack}}_s \iff \)  
- **Closure Operator Semantics**: All the fixed points of \( f \) take the form \( w.\text{false}^\omega \iff \)  
- **FLTL Characterization**: \( \text{TL[}[P] |\models_T \diamond \text{attack} \).
Language for Structured Communication (sessions)

Definition (The HVK language (Honda 98))

\[ P, Q ::= \]

- \texttt{request} \(a(k)\) \texttt{in} \(P\) \hspace{1cm} \text{Session Request}
- \texttt{accept} \(a(x)\) \texttt{in} \(P\) \hspace{1cm} \text{Session Acceptance}
- \(k!\vec{e}; P\) \hspace{1cm} \text{Data Sending}
- \(k?(x)\) \texttt{in} \(P\) \hspace{1cm} \text{Data Reception}
- \(k \triangleleft l; P\) \hspace{1cm} \text{Label Selection}
- \(k \triangleright \{l_1 : P_1 \parallel \cdots \parallel l_n : P_n\}\) \hspace{1cm} \text{Label Branching}

Further guarantees are needed when dealing with sessions, e.g.:

- Sessions should be of finite time.
- One should be able to cancel a session.

HVK-T

HVK

+ session cancellation and constraint-guarded accepts.

P ::= \texttt{request} \(a(k)\) \texttt{during} \(m\) \texttt{in} \(P\) \hspace{1cm} \text{Timed Session Request}
- \texttt{accept} \(a(k)\) \texttt{given} \(c\) \texttt{in} \(P\) \hspace{1cm} \text{Declarative Session Acceptance}
- \(\cdots\)
- \texttt{kill} \(c\) \hspace{1cm} \text{Session Abortion}
Language for Structured Communication (sessions)

Definition (The HVK language (Honda 98) )

\[
P, Q ::= \text{request } a(k) \text{ in } P \quad \text{Session Request}
\]
\[
\text{accept } a(x) \text{ in } P \quad \text{Session Acceptance}
\]
\[
k!\tilde{e}; P \quad \text{Data Sending}
\]
\[
k?(x) \text{ in } P \quad \text{Data Reception}
\]
\[
k \triangleright \{l_1 : P_1 \parallel \cdots \parallel l_n : P_n\} \quad \text{Label Branching}
\]

Further guarantees are needed when dealing with sessions, e.g.:

- Sessions should be of finite time.
- One should be able to cancel a session.
Language for Structured Communication (sessions)

Definition (The HVK language (Honda 98) )

\[ P, Q ::= \begin{align*}
    & \text{request } a(k) \text{ in } P & \text{Session Request} \\
    & \text{accept } a(x) \text{ in } P & \text{Session Acceptance} \\
    & k!\vec{e}; P & \text{Data Sending} \\
    & k?(x) \text{ in } P & \text{Data Reception} \\
    & k \triangleq l; P & \text{Label Selection} \\
    & k \triangleright \{ l_1 : P_1 \parallel \cdots \parallel l_n : P_n \} & \text{Label Branching}
\end{align*} \]

Further guarantees are needed when dealing with sessions, e.g.:

- Sessions should be of finite time.
- One should be able to cancel a session.

HVK-T

\[ P ::= \begin{align*}
    & \text{request } a(k) \text{ during } m \text{ in } P & \text{Timed Session Request} \\
    & \text{accept } a(k) \text{ given } c \text{ in } P & \text{Declarative Session Acceptance} \\
    & \ldots & \\
    & \text{kill } c_k & \text{Session Abortion}
\end{align*} \]
Dynamic Interactive Scores

Dynamic Reconfiguration when Interacting:

- Moving boxes.
- Adding/deleting intervals.

Verification: Minimal conditions to avoid raise conditions.
Outline

1. Intuitive Description and SOS
2. Symbolic Semantics
3. Undecidability of FLTL
4. Denotational Semantics for \texttt{utcc}
5. Applications
6. Abstract Semantics and Static Analysis of \texttt{utcc}
7. Concluding Remarks
Abstract Interpretation of utcc programs

- The behavior of a program $P$ can be approximated by computing $\llbracket P \rrbracket^\tau_{\alpha}$, a compact representation of $\llbracket P \rrbracket$. 
Abstract Interpretation of utcc programs

- The behavior of a program $P$ can be approximated by computing $\llbracket P \rrbracket_{\alpha}^\tau$, a compact representation of $\llbracket P \rrbracket$.

Two steps:

Abstracting the constraint system $(C, \alpha, A)$

- To reuse previously defined abstract domains for logic programming (e.g. groundness analysis, types, etc).
- To bound the behavior of the operator $(\text{abs } \vec{x}; c) P$. 
Abstract Interpretation of utcc programs

- The behavior of a program $P$ can be approximated by computing $\lbrack P \rbrack^\tau_\alpha$, a compact representation of $\lbrack P \rbrack$.

Two steps:

Abstracting the constraint system $(C, \alpha, A)$

- To reuse previously defined abstract domains for logic programming (e.g. groundness analysis, types, etc).
- To bound the behavior of the operator $(\texttt{abs } \vec{x}; c) P$.

Abstracting the sequences $(\tau)$

- To obtain a finite cut approximating the infinite sequences of the concrete semantics.
Applications

- **Groundness Analysis** of utcc programs. We reuse two abstract domains from logic programming:
  - Pos [AMSS98]: Positive propositional formulae to represent groundness dependencies. E.g., $\alpha_g(x = [y|z]) = \text{iff}(x, \{y, z\})$
  - Type dependencies [CSS99]: e.g., $\alpha(x = [a|y]) = \text{list}(x, y)$

- **Secrecy Analysis**: Depth-$\kappa$ cut to approximate the behavior of the protocol:

\[
cut_{\kappa}(m) = \begin{cases} 
  m & \text{if } \text{length}(m) \leq \kappa \\
  m_T & \text{otherwise}
\end{cases}
\]
Concluding Remarks

- We proposed utcc, a declarative model for the specification of mobile reactive systems.
- **Reasoning techniques for utcc:**
  - Operational and symbolic semantics.
  - Semantics based on closure operators.
  - Declarative interpretation of processes as formulae in FLTL.
  - Abstract semantics for the static analysis of utcc programs.
- We showed the applicability of utcc in several domains:
  - Decidability of Pnueli’s FLTL.
  - Analysis of security protocols.
  - Declarative interpretation of sessions.
  - Modeling of Multimedia Interactive Systems.
Concluding Remarks

Publications from this dissertation

Concluding Remarks

Future Work

- Non-deterministic / probabilistic choices for modeling purposes.
- Proof system in the lines of [NPV02].
- Studying the minimal number of global variables required to obtain undecidability in Monadic FLTL [DFL02].
- Abstract debugging of utcc programs.
- Analysis of properties related to mobile systems (e.g. [Fer05]).
- Automatic verification of tcc and utcc.
Thank you!
Two classes of Boolean functions for dependency analysis.

J.A. Bergstra and J.W. Klop.
Algebra of communicating processes with abstraction.

M. Codish, H. Søndergaard, and P. Stuckey.
Sharing and groundness dependencies in logic programs.

Anatoli Degtyarev, Michael Fisher, and Alexei Lisitsa.
Equality and monodic first-order temporal logic.
Studia Logica, 72(2), 2002.

Jérôme Feret.
Abstract interpretation of mobile systems.

Francois Fages, Paul Ruet, and Sylvain Soliman.
Linear concurrent constraint programming: Operational and phase semantics.


C. A. R. Hoare.

*Communications Sequential Processes*.

Cosimo Laneve and Ugo Montanari.
Mobility in the CC-paradigm.

Stephan Merz.
Decidability and incompleteness results for first-order temporal logics of linear time.


R. Milner.

*Communication and Concurrency*.

SU Fisher Research 511/24.
R. Milner, J. Parrow, and D. Walker.  
A calculus of mobile processes, Parts I and II.  

M. Nielsen, C. Palamidessi, and F.D. Valencia.  
Temporal concurrent constraint programming: Denotation, logic and applications.  

Vijay A. Saraswat.  
*Concurrent Constraint Programming*.  

Vijay Saraswat, Radha Jagadeesan, and Vineet Gupta.  
Foundations of timed concurrent constraint programming.  

Vijay Saraswat and Patrick Lincoln.  
Higher-order Linear Concurrent Constraint Programming.  
Frank D. Valencia.
Decidability of infinite-state timed ccp processes and first-order ltl.
Approaches to mobility in CCP

1. [Sar93] Suppose a injective function $f$. Let $P = \exists z \text{tell}(x = f(z))$ and $Q = \exists y (\text{ask } x = f(y) \text{ then tell}(x = f(y)) \parallel R)$. **Drawbacks:** If two names are sent on $x$ those names must be equals (otherwise an inconsistency arises).

2. [LM92] Let $P = \exists z \text{tell}(x =< 0, z >)$ and $Q = \exists y, y' (\text{ask } x =< y, y' > \text{ then tell}(x =< y, y' >) \rightarrow R)$. Here the **atomic tell** avoids inconsistencies when two messages are sent on $x$. **Drawbacks:** Atomic tells introduce non-determinism loosing the strong connection CCP has with logic.
A constraint system is a tuple $\langle \sum, \Delta \rangle$ where $\sum$ is a signature and $\Delta$ a consistent first-order theory over $\sum$.

Constraints are first-order formulae over $\sum$.

Entailment relation $c \models d$ holds iff $c \Rightarrow d$ is valid on $\Delta$. $c \equiv d$ iff $c \models d$ and $d \models c$.

$C$ denotes the set of constraints modulo $\equiv$ in $\langle \sum, \Delta \rangle$. 

Concluding Remarks
Let $\equiv$ be the smallest congruence satisfying:

1. $P \equiv Q$ if they differ only by a renaming of bound variables (alpha-conversion).
2. $P \parallel \text{skip} \equiv P$
3. $P \parallel Q \equiv Q \parallel P$
4. $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$
5. $P \parallel (\text{local } \vec{x}; c) Q \equiv (\text{local } \vec{x}; c) (P \parallel Q)$ if $\vec{x} \notin \text{fv}(P)$ (Scope Extrusion)
6. $(\text{local } \vec{x}; c) (\text{local } \vec{y}; d) P \equiv (\text{local } \vec{x}; \vec{y} ; c \land d) P$ if $\vec{x} \cap \vec{y} = \emptyset$ and $\vec{y} \notin \text{fv}(c)$. 

Concluding Remarks
Operational Semantics

Internal Reductions

**R\_TELL**
\[
\langle \text{tell}(c), d \rangle \rightarrow \langle \text{skip}, d \land c \rangle
\]

**R\_PAR**
\[
\langle P, c \rangle \rightarrow \langle P', d \rangle
\]
\[
\langle P \parallel Q, c \rangle \rightarrow \langle P' \parallel Q, d \rangle
\]

**R\_LOC**
\[
\langle P, c \land (\exists \vec{x} d) \rangle \rightarrow \langle P', c' \land (\exists \vec{x} d) \rangle
\]
\[
\langle \text{local} \ \vec{x}; c \rangle P, d \rightarrow \langle \text{local} \ \vec{x}; c' \rangle P', d \land \exists \vec{x} c'
\]

**R\_UNL**
\[
d \models_\Delta c
\]
\[
\langle \text{unless} \ c \ \text{next} \ P, d \rangle \rightarrow \langle \text{skip}, d \rangle
\]

**R\_REP**
\[
\langle \text{!} P, d \rangle \rightarrow \langle P \parallel \text{next} \ \text{!} P, d \rangle
\]

**R\_ABS**
\[
d \models_\Delta c[\vec{t}/\vec{x}] \quad [\vec{t}/\vec{x}] \text{ is admissible.}
\]
\[
\langle \text{abs} \ \vec{x}; c \rangle P, d \rightarrow \langle P[\vec{t}/\vec{x}] \parallel (\text{abs} \ \vec{x}; c \land \vec{x} \neq \vec{t}) P, d \rangle
\]

**R\_STR**
\[
\frac{\gamma_1 \rightarrow \gamma_2}{\gamma_1' \rightarrow \gamma_2'} \quad \text{if } \gamma_1 \equiv \gamma_1' \text{ and } \gamma_2 \equiv \gamma_2'
\]
Operational Semantics

Observable Transition

\[
R_{\text{OBS}} \quad \frac{\langle P, c \rangle \longrightarrow^* \langle Q, d \rangle \not\rightarrow}{P \overset{(c,d)}{\longrightarrow} F(Q)}
\]

Future Function

Let \( F \) be a partial function defined as:

\[
F(P) = \begin{cases} 
\text{skip} & \text{if } P = \text{skip} \\
\text{skip} & \text{if } P = (\text{abs } \vec{x}; c) Q \\
F(P_1) \parallel F(P_2) & \text{if } P = P_1 \parallel P_2 \\
(\text{local } \vec{x}) F(Q) & \text{if } P = (\text{local } \vec{x}; c) Q \\
Q & \text{if } P = \text{next } Q \\
Q & \text{if } P = \text{unless } c \text{ next } Q
\end{cases}
\]
Theorem (Semantic Correspondence)

Let $P$ be an abstracted-unless free process. Suppose that

\[ P \xrightarrow{(c_1,d_1)} P_1 \xrightarrow{(c_2,d_2)} \ldots \xrightarrow{(c_i,d_i)} P_i \text{ and} \]

\[ P \xrightarrow{(c_1,e_1)} P_1' \xrightarrow{(c_2,e_2)} \ldots \xrightarrow{(c_i,e_i)} P_i'. \]

Then for every $c \in C$ and $j \in \{1, \ldots, i\}$, $d_i \models c$ iff $e_i \models_T c$. 

Concluding Remarks
Expressiveness of \texttt{utcc}

Minsky Machines

Instructions using two counters $c_0$ and $c_1$:

- $L_i: \text{HALT}$
- $L_i: c_n := c_n + 1; \text{goto } L_j$
- $L_i: \text{if } c_n = 0 \text{ then goto } L_j \text{ else } c_n := c_n - 1; \text{goto } L_k$

The machine $M$:

1. **Starts** at instruction $L_1$ with $c_0 = c_1 = 0$.
2. **Halts** if the control reaches the location of a halt instruction.
Encoding of Minsky Machines into utcc

\[
\begin{align*}
\text{ZERO}_n & \overset{\text{def}}{=} \text{when } \text{inc}_n \text{ do next (local } a) (\text{NOT-ZERO}_n(a) \parallel \text{!when out}(a) \text{ do ZERO}_n) \parallel \\
& \quad \text{when } \text{idle}_n \text{ do next ZERO}_n \parallel \\
& \quad \text{tell(isz}_n) \\
\text{NOT-ZERO}_n(x) & \overset{\text{def}}{=} \text{when } \text{inc}_n \text{ do next (local } b) (\text{NOT-ZERO}_n(b) \parallel \text{!when out}(b) \text{ do NOT-ZERO}_n(x)) \parallel \\
& \quad \text{when } \text{dec}_n \text{ do next tell(out}(x)) \parallel \\
& \quad \text{when } \text{idle}_n \text{ do next NOT-ZERO}_n(x) \parallel \\
& \quad \text{tell(not-zero}_n)
\end{align*}
\]

\[
\begin{align*}
\text{ins}(l_i, \text{HALT}) & = \text{tell(halt)} \parallel \text{next tell(out}(l_i)) \parallel \text{tell(idle}_0 \land \text{idle}_1) \\
\text{ins}(l_i, \text{INC}(c_n, l_j)) & = \text{tell(inc}_n) \parallel \text{next tell(out}(l_j)) \parallel \text{tell(idle}_{1-n}) \\
\text{ins}(l_i, \text{DECJ}(c_n, l_j, l_k)) & = \text{when } \text{isz}_n \text{ do (next tell(out}(l_j)) \parallel \text{tell(idle}_n)) \parallel \\
& \quad \text{when } \text{not-zero}_n \text{ do (tell(dec}_n) \parallel \text{next tell(out}(l_k))) \parallel \\
& \quad \text{tell(idle}_{1-n})
\end{align*}
\]
Strongest Postcondition and Fixed Formulae

**Strongest Postcondition**

The *Strongest Postcondition* of *P*, denoted by $sp_s(P)$, is defined as the set \( \{ w \mid P \xrightarrow{(w,v)} s \text{ and } w \in Fix(v) \} \).

**Definition (Fixed Formulae)**

Let \( n \geq 0 \) and \( Fix : FF \rightarrow \mathcal{P}(FF) \) be defined as

- \( Fix(c) = \{ F \in FF \mid F \models_T c \} \)
- \( Fix(F_1 \land F_2) = \{ F \in FF \mid F \in Fix(F_1) \text{ and } F \in Fix(F_2) \} \)
- \( Fix(\forall \vec{x} \oplus^n (c) \Rightarrow F_1) = \{ F \in FF \mid \text{for all } \vec{x}\text{-variant } F' \text{ of } F, \text{ if } F' \models_T (\oplus^n(c) \land \vec{x} = \vec{t}) \text{ for some } \vec{t} \in T \text{ s.t. } \text{adm}(\vec{x}, \vec{t}) \text{ then } F' \in Fix(F_1) \} \)
- \( Fix(\exists \vec{x} F_1) = \{ F \in FF \mid \text{there exists an } \vec{x}\text{-variant } F' \text{ of } F \text{ s.t. } F' \in Fix(F_1) \} \)
- \( Fix(\ominus F_1) = \{ F \in FF \mid F = \ominus F' \text{ and } F' \in Fix(F_1) \} \)

Given the future-free formulae \( F \) and \( G \), if \( F \in Fix(G) \) we say that \( F \) is a fixed formula for \( G \).
A **Constraint System** defines the basic constraints agents can tell or ask. A cylindric c.s. is a structure \( C = \langle C, \leq, \sqcup, \text{true}, \text{false}, \text{Var}, \exists, d \rangle \) s.t.:

- \( \langle C, \leq, \sqcup, \text{true}, \text{false} \rangle \) is a lattice.
- \( \text{Var} \) is a denumerable set of variables.
- \( \exists_x : C \to C \) is a cylindrification operator helpful to define the local (hiding) operator.
- For each \( x, y \in \text{Var}, d_{xy} \in C \) is a diagonal element (e.g. \( x = y \)) to model parameter passing.
### Semantic Equations

<table>
<thead>
<tr>
<th>Name</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>D_SKIP</strong></td>
<td>$[[\text{skip}]]_I = \mathcal{C}^\omega$</td>
</tr>
<tr>
<td><strong>D_TELL</strong></td>
<td>$[[\text{tell}(c)]]_I = {d.s \in \mathcal{C}^\omega \mid d \models c}$</td>
</tr>
<tr>
<td><strong>D_PAR</strong></td>
<td>$[[P \parallel Q]]_I = [[P]]_I \cap [[Q]]_I$</td>
</tr>
<tr>
<td><strong>D_NEXT</strong></td>
<td>$[[\text{next } P]]_I = {d.s \in \mathcal{C}^\omega \mid s \in [[P]]_I}$</td>
</tr>
<tr>
<td><strong>D_UNL</strong></td>
<td>$[[\text{unless } c \text{ next } P]]_I = {d.s \in \mathcal{C}^\omega \mid d \notmodels c \text{ and } s \in [[P]]_I}$ $\cup {d.s \in \mathcal{C}^\omega \mid d \models c}$</td>
</tr>
<tr>
<td><strong>D_REP</strong></td>
<td>$[[! P]]_I = {s \in \mathcal{C}^\omega \mid \text{for all } w, s' \text{ st } s = w.s' \text{ and } s' \in [[P]]_I}$</td>
</tr>
<tr>
<td><strong>D_LOC</strong></td>
<td>$[[\text{(local } \vec{x}; c) P]]_I = {s \in \mathcal{C}^\omega \mid \text{there exists an } \vec{x}-\text{variant } s' \text{ of } s \text{ st }$ $s'(1) \models c \text{ and } s' \in [[P]]_I}$</td>
</tr>
<tr>
<td><strong>D'ABS</strong></td>
<td>$[[\text{when } c \text{ do } P]]_I = {d.s \in \mathcal{C}^\omega \mid d \models c \text{ and } d.s \in [[P]]_I}$ $\cup {d.s \in \mathcal{C}^\omega \mid d \notmodels c}$</td>
</tr>
<tr>
<td><strong>D_ABS</strong></td>
<td>$[[\text{(abs } \vec{x}; c) P]]<em>I = \bigcap</em>{\vec{t} \in T^{</td>
</tr>
<tr>
<td><strong>D_CALL</strong></td>
<td>$[[p(\vec{x})]]_I = I(p(\vec{x}))$</td>
</tr>
</tbody>
</table>
# Abs. Semantics Equations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abskip</td>
<td>( [\text{skip}]^T_X = \tau(\mathcal{A}^\omega) )</td>
</tr>
<tr>
<td>Atell</td>
<td>( [\text{tell}(c)]^T_X = \tau({d_{\kappa}.s_{\kappa} \in \mathcal{A}^\omega \mid d_{\kappa} \vdash_{\alpha} c}) )</td>
</tr>
<tr>
<td>Apar</td>
<td>( [P \parallel Q]^T_X = [P]^T_X \cap [Q]^T_X )</td>
</tr>
<tr>
<td>Anext</td>
<td>( [\text{next } P]^T_X = \tau({d_{\kappa}.s_{\kappa} \in \mathcal{A}^* \mid s_{\kappa} \in [P]^T_X}) )</td>
</tr>
<tr>
<td>Aunl</td>
<td>( [\text{unless } c \text{ next } P]^T_X = \tau(\mathcal{A}^\omega) )</td>
</tr>
<tr>
<td>Arep</td>
<td>( ![P]^T_X = \tau({s_{\kappa} \in \mathcal{A}^* \mid \text{for all } s'<em>{\kappa}, w</em>{\kappa} \text{ s.t.} s_{\kappa} = w_{\kappa}.s'<em>{\kappa}, s'</em>{\kappa} \in [P]^T_X}) )</td>
</tr>
<tr>
<td>Aloc</td>
<td>( [(\text{local } \vec{x}; c) P]^T_X = \tau({s_{\kappa} \in \mathcal{A}^* \mid \text{there exists a } \vec{x}\text{-variant } s'<em>{\kappa} \text{ of } s</em>{\kappa} \text{ s.t. } s'<em>{\kappa}(1) \vdash</em>{\alpha} c \text{ and } s'_{\kappa} \in [P]^T_X}) )</td>
</tr>
<tr>
<td>Aabs</td>
<td>( [\text{when } c \text{ do } P]^T_X = \tau({d_{\kappa}.s_{\kappa} \in \mathcal{A}^\omega \mid d_{\kappa} \not\vdash_{\mathcal{A}} c} \cup {d_{\kappa}.s_{\kappa} \in \mathcal{A}^* \mid d_{\kappa} \vdash_{\mathcal{A}} c \text{ and } d_{\kappa}.s_{\kappa} \in [P]^T_X}) )</td>
</tr>
<tr>
<td>Aabs</td>
<td>( [\text{(abs } \vec{x}; c) P]^T_X = \bigcap_{t_{\kappa} \in T_{\kappa}</td>
</tr>
<tr>
<td>Acall</td>
<td>( [p(\vec{x})]^T_X = X(p(\vec{x})) )</td>
</tr>
</tbody>
</table>
DS Attack

\begin{align*}
msg_1 & : A \rightarrow C : \{A, m\}_{pub(C)} \\
msg'_1 & : C \rightarrow B : \{A, m\}_{pub(B)} \\
msg_2 & : B \rightarrow A : \{n\}_{pub(m)}
\end{align*}
Abstract Semantics

Abstract Constraint Systems

Let $C$ and $A$ be constraint systems. A description $(C, \alpha, A)$ consists of an abstract domain $(A, \leq^\alpha)$ and a monotonic abstraction function $\alpha : C \rightarrow A$. 
Abstract Semantics

Abstract Constraint Systems

Let $C$ and $A$ be constraint systems. A description $(C, \alpha, A)$ consists of an abstract domain $(A, \leq^\alpha)$ and a monotonic abstraction function $\alpha : C \rightarrow A$.

Sequence Abstraction

$\tau : (A^\omega \cup A^*) \rightarrow A^*$ is a reductive operator $(\tau(s_K) \leq^\alpha s_K)$. 
Abstract Semantics

Semantic Equations

\[ \begin{align*}
A_{\text{TELL}} \quad [\text{tell}(c)]^\tau_X &= \tau(\{d_{\kappa}.s_{\kappa} \in \mathcal{A}^{\omega} \mid d_{\kappa} \vdash^\alpha \alpha(c)\}) \\
A'_{\text{ABS}} \quad [\text{when } c \text{ do } P]^\tau_X &= \tau(\{d_{\kappa}.s_{\kappa} \in \mathcal{A}^{\omega} \mid d_{\kappa} \not\vdash \mathcal{A}c\}) \\
&\quad \cup \{d_{\kappa}.s_{\kappa} \in \mathcal{A}^* \mid d_{\kappa} \vdash \mathcal{A}c \text{ and } d_{\kappa}.s_{\kappa} \in \lbrack P \rbrack^\tau_X\} \\
A_{\text{CALL}} \quad [p(\vec{x})]^\tau_X &= \chi(p(\vec{x}))
\end{align*} \]

The Abs. semantics is defined as the lfp of

\[ T^\alpha_D(\chi)(p(\vec{x})) = \lbrack (\Delta^\vec{y}_X P) \rbrack^\tau_X \text{ if } p(\vec{y}) \overset{\text{def}}{=} P \in D \]
Abstract Semantics

Semantic Equations

\[ A_{\text{TELL}} \quad \llbracket \text{tell}(c) \rrbracket^\tau_X = \tau(\{ d_{\kappa} \cdot s_{\kappa} \in A^\omega \mid d_{\kappa} \forces_\alpha \alpha(c) \}) \]

\[ A'_{\text{ABS}} \quad \llbracket \text{when } c \text{ do } P \rrbracket^\tau_X = \tau(\{ d_{\kappa} \cdot s_{\kappa} \in A^\omega \mid d_{\kappa} \not\models A \check{c} \}) \]
\[ \quad \cup \{ d_{\kappa} \cdot s_{\kappa} \in A^* \mid d_{\kappa} \models A \check{c} \text{ and } d_{\kappa} \cdot s_{\kappa} \in \llbracket P \rrbracket^\tau_X \} \]

\[ A_{\text{CALL}} \quad \llbracket p(\vec{x}) \rrbracket^\tau_X = X(p(\vec{x})) \]

Abstract Semantics

The Abs. semantics is defined as the lfp of

\[ T^\alpha_D(X)(p(\vec{x})) = \llbracket (\Delta^\vec{y}_X P) \rrbracket^\tau_X \text{ if } p(\vec{y}) \overset{\text{def}}{=} P \in D \]

Theorem (Soundness of the approximation)

Given a utcc program \( D \cdot P \), if \( s \in \llbracket P \rrbracket \) then \( \tau(\alpha(s)) \in \llbracket P \rrbracket^\tau \).
Correctness of the Abstraction

Approximation: Let \( d_\kappa = \alpha(d) \). We say that \( d_\kappa \) is the best approximation of \( d \). Furthermore, for all \( c_\kappa \leq^\alpha d_\kappa \) we say that \( c_\kappa \) approximates \( d \) and we write \( c_\kappa \propto d \).

Correctness

Let \( \alpha : C \to A \) be monotone. We say that \( A \) is upper correct w.r.t \( C \) if for all \( c \in C \) and \( x, y \in V \):

1. \( \alpha(\exists_x c) = \exists^\alpha_x \alpha(c) \).
2. \( \alpha(d_{xy}) = d_{xy}^{\alpha} \). And
3. \( \alpha(c \uplus d) \models^\alpha \alpha(c) \uplus^\alpha \alpha(d) \).

Let \( \alpha_t \) be the induced term-abstraction by \( \alpha \). Given the sequence of variables \( \vec{x} \) and \( \vec{t}, \vec{t}' \in T_{|\vec{x}|} \), 4) if \( \alpha_t(\vec{t}) = \alpha_t(\vec{t}') \) then \( \alpha(c[\vec{t}/\vec{x}]) = \alpha(c[\vec{t}'/\vec{x}]) \).
Soundness of the Abstraction

Abstract domain

$\mathbb{A} = (A, \subseteq^\alpha)$ where $A = \mathcal{P}(A^*)$ and $\subseteq^\alpha$ is defined similarly to $\subseteq^c$ (Smyth powerdomain). We require $\mathbb{A}$ to be noetherian.
Soundness of the Abstraction

Abstract domain

\( \mathbb{A} = (A, \subseteq^\alpha) \) where \( A = \mathcal{P}(A^*) \) and \( \subseteq^\alpha \) is defined similarly to \( \subseteq^c \) (Smyth powerdomain). We require \( \mathbb{A} \) to be noetherian.

Galois Connection

\[
\alpha(E) := \tau(\{\alpha'(s) \mid s \in E\}) \\
\gamma(A) := \{s \mid \tau(\alpha'(s)) \in A\}
\]
Soundness of the Abstraction

Abstract domain

\[ \mathbb{A} = (A, \subseteq^\alpha) \text{ where } A = \mathcal{P}(A^*) \text{ and } \subseteq^\alpha \text{ is defined similarly to } \subseteq^c \text{ (Smyth powerdomain).} \text{ We require } \mathbb{A} \text{ to be noetherian} \]

Galois Connection

\[ \begin{align*}
\alpha(E) & := \tau(\{ \alpha'(s) \mid s \in E \}) \\
\gamma(A) & := \{ s \mid \tau(\alpha'(s)) \in A \}
\end{align*} \]

Let \( I : \text{ProcHeads} \rightarrow E \) (an interpretation ), \( X : \text{ProcHeads} \rightarrow A \) (abstract interpretation ) and \( p \) a procedure name. Then

\[ \begin{align*}
\alpha(I)(p) & := \tau(\{ \alpha'(s) \mid s \in I(p) \}) \\
\gamma(X)(p) & := \{ s \mid \tau(\alpha'(s)) \in X(p) \}
\end{align*} \]