Universal Temporal Concurrent Constraint Programming.

Carlos Olarte's Ph.D Defense.
Supervisors: C. Palamidessi and F. Valencia.

INRIA and LIX, École Polytechnique Paris.

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.

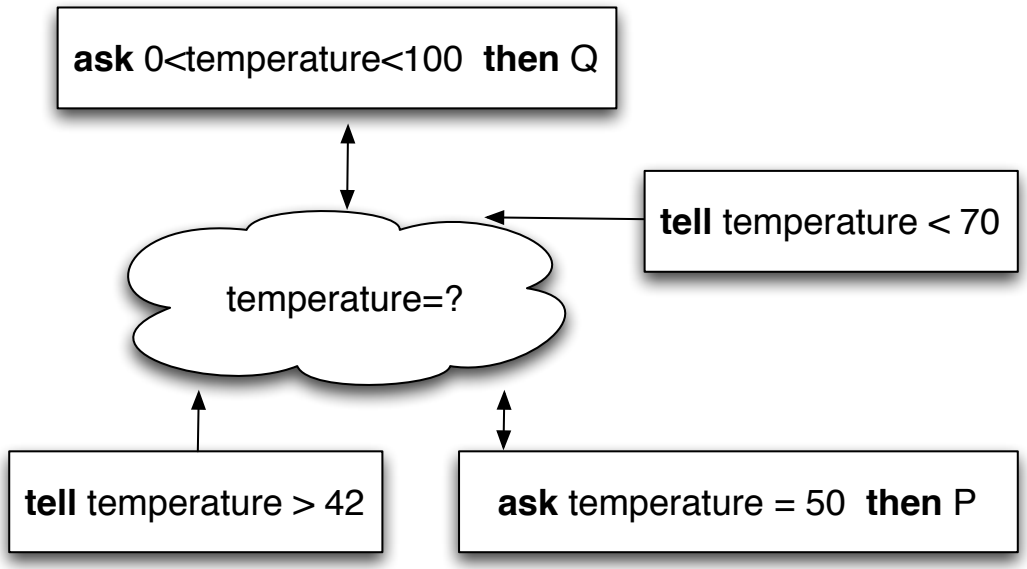
Some Examples : CCS [Mil89], the π-calculus [MPW92], CSP [Hoa85], ACP [BK85], **CCP** [Sar93].

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. $x > 42 \models x > 0$).

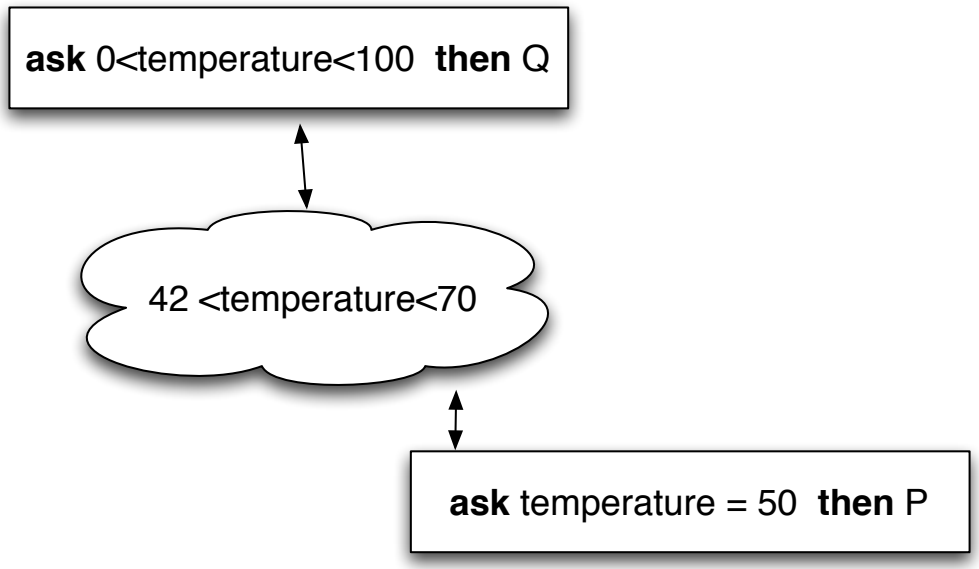
Motivation

- 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. $x > 42 \models x > 0$).



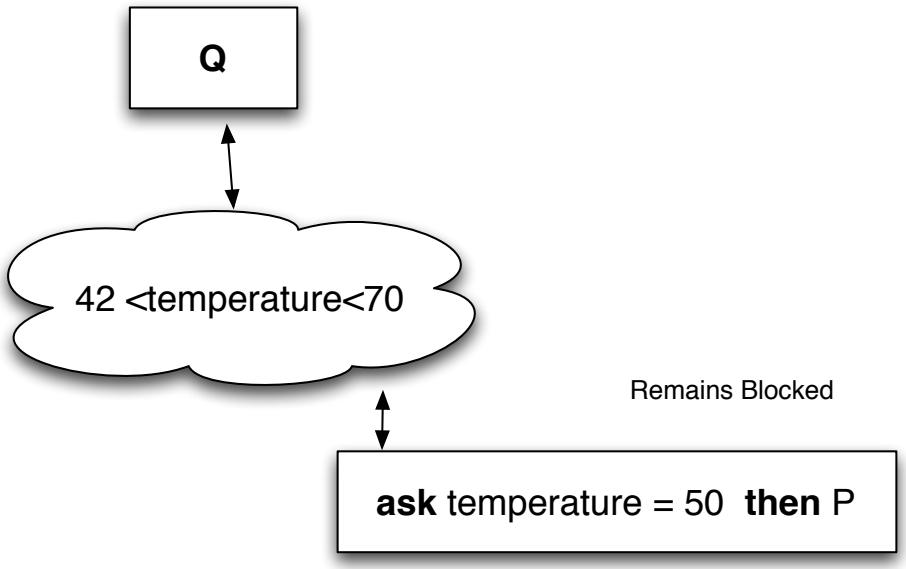
Motivation

- 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. $x > 42 \models x > 0$).



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 rechability analysis using deduction in logic.
- Determinism : which is the source of CCP's elegant and simple semantic characterizations.
- Applications in emergent application areas.

Our Approach

Mobility in CCP

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.

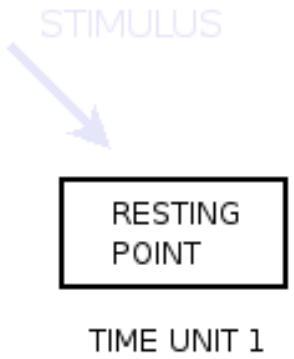
Outline

Outline

The tcc Model [SJG94]

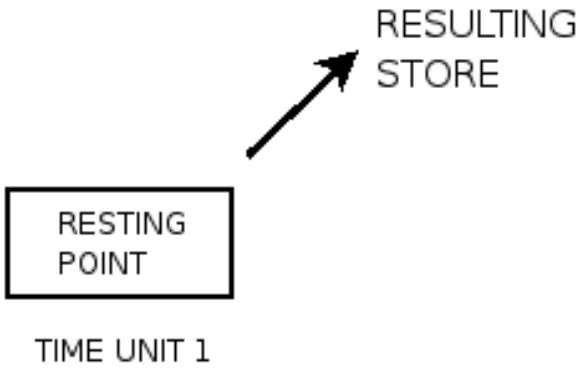
1. Receives a stimulus (i.e a constraint) from the environment.

The tcc Model [SJG94]



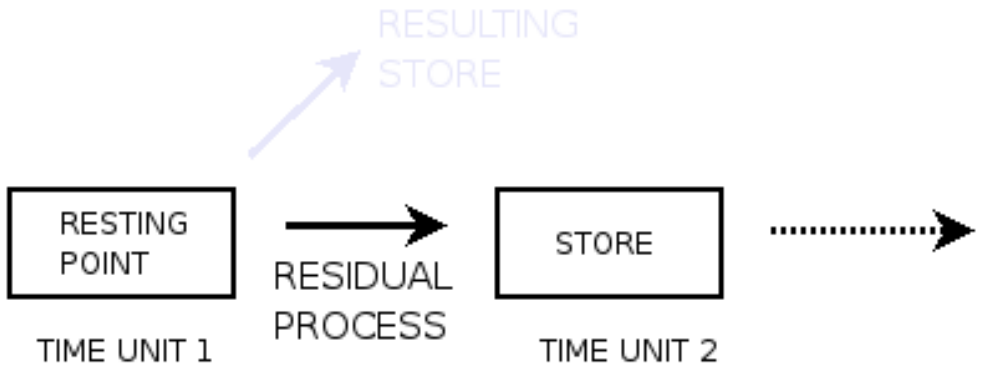
1. Receives a stimulus (i.e a constraint) from the environment.
2. Computes a CCP process in the current time-unit and wait for stability.

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

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

- $\textbf{tell}(c)$: adds c to the store in the current time interval.
- $\textbf{when } c \textbf{ do } P$: executes P if c can be deduced from the current store.
- $(\textbf{local } \vec{x})\, P$: behaves like P but the information about variables in $\vec{x}$ is local to P
- $\textbf{next } P$: executes P in the next time unit.
- $\textbf{unless } c \textbf{ next } P$: 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 $Q = \textbf{tell}(\mathtt{out}(42))$ and $P = \textbf{when}\ \ \mathtt{out}(x)\ \textbf{do}\ R$.

- x is not a place holder in P
- $\mathtt{out}(42) \not\models \mathtt{out}(x)$. Then $P \parallel Q \not\longrightarrow$.

Abstractions and the utcc Calculus

Example

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

- x is not a place holder in P
- $\mathtt{out}(42) \not\models \mathtt{out}(x)$. Then $P \parallel Q \not\longrightarrow$.

The idea of Abstractions in utcc

The abstraction construct $S = (\textbf{abs}\ \vec{x}; c)\, R$:

- S can be seen as a λ-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., $S \parallel Q \longrightarrow R[42/x]$.
- 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 ($\longrightarrow$)

$$\mathrm{R}_{\mathrm{TELL}}\ \frac{}{\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{skip}, d \wedge c\rangle}$$

$$\mathrm{R}_{\mathrm{PAR}}\ \frac{\langle P, c\rangle \longrightarrow \langle P', d\rangle}{\langle P \parallel Q, c\rangle \longrightarrow \langle P' \parallel Q, d\rangle}$$

$$\mathrm{R}_{\mathrm{ABS}}\ \frac{d \models_{\Delta} c[\vec{t}/\vec{x}] \qquad [\vec{t}/\vec{x}] \text{ is admissible.}}{\langle (\mathbf{abs}\ \vec{x}; c)\, P, d\rangle \longrightarrow \left\langle P[\vec{t}/\vec{x}] \parallel (\mathbf{abs}\ \vec{x}; c \wedge \vec{x} \neq \vec{t}\ )\, P, d\right\rangle}$$

Observable transitions ($\Longrightarrow$)

$$\mathrm{R}_{\mathrm{OBS}}\ \frac{\langle P, c\rangle \longrightarrow^{*} \langle Q, d\rangle \not\longrightarrow}{P \overset{(c,d)}{\Longrightarrow} F(Q)}$$

$$\begin{cases} F((\mathbf{abs}\ \vec{x}; c)\, Q) = \mathbf{skip} \\ F(\mathbf{next}\, Q) = Q \\ F(\mathbf{unless}\ c\ \mathbf{next}\, Q) = Q \end{cases}$$

▸ C. System ▸ SOS Details

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 $P = P_1 \xrightarrow{(c_1,c'_1)} P_2 \xrightarrow{(c_2,c'_2)} ...P_i \xrightarrow{(c_i,c'_i)} ....$

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

Some Examples

Communication using global channels

$$Alice \xrightarrow{\{m\}_B} Bob$$

$$\begin{array}{lcl} A & = & (\mathbf{local}\, m)\,(\mathbf{tell}(\mathtt{out}(\{m\}_B))) \\ B & = & (\mathbf{abs}\; x; \mathtt{out}(\{x\}_B))\, B' \end{array}$$

Some Examples

Communication using global channels

$$Alice \xrightarrow{\{m\}_B} Bob$$

$$\begin{array}{rcl} A & = & (\textbf{local}\, m)\,(\textbf{tell}(\mathtt{out}(\{m\}_B))) \\ B & = & (\textbf{abs}\; x; \mathtt{out}(\{x\}_B))\, B' \end{array}$$

	Process	*Store*
•	$A \parallel B$	`true`

Some Examples

Communication using global channels

$$Alice \xrightarrow{\{m\}_B} Bob$$

$$\begin{array}{rcl} A & = & (\textbf{local}\, m)\,(\textbf{tell}(\mathtt{out}(\{m\}_B))) \\ B & = & (\textbf{abs}\; x; \mathtt{out}(\{x\}_B))\, B' \end{array}$$

	Process	*Store*
•	$A \parallel B$	`true`
•	$(\textbf{local}\, m)\,(\textbf{tell}(\mathtt{out}(\{m\}_B)) \parallel B)$	`true`

Some Examples

Communication using global channels

$$Alice \xrightarrow{\{m\}_B} Bob$$

$$\begin{array}{rcl} A & = & (\textbf{local}\, m)\,(\textbf{tell}(\mathtt{out}(\{m\}_B))) \\ B & = & (\textbf{abs}\; x; \mathtt{out}(\{x\}_B))\, B' \end{array}$$

	Process	*Store*
•	$A \parallel B$	$\mathtt{true}$
•	$(\textbf{local}\, m)\,(\textbf{tell}(\mathtt{out}(\{m\}_B)) \parallel B)$	$\mathtt{true}$
•	$(\textbf{local}\, m; \mathtt{out}(\{m\}_B))\,(B)$	$\exists m.(\mathtt{out}(\{m\}_B))$

Some Examples

Communication using global channels

$$Alice \xrightarrow{\{m\}_B} Bob$$

$$\begin{array}{rcl} A & = & (\textbf{local}\, m)\,(\textbf{tell}(\mathtt{out}(\{m\}_B))) \\ B & = & (\textbf{abs}\; x; \mathtt{out}(\{x\}_B))\, B' \end{array}$$

	Process	*Store*
•	$A \parallel B$	$\mathtt{true}$
•	$(\textbf{local}\, m)\,(\textbf{tell}(\mathtt{out}(\{m\}_B)) \parallel B)$	$\mathtt{true}$
•	$(\textbf{local}\, m; \mathtt{out}(\{m\}_B))\,(B)$	$\exists m.(\mathtt{out}(\{m\}_B))$
•	$(\textbf{local}\, m; \mathtt{out}(\{m\}_B))\,(B'[m/x] \parallel B'')$	$\exists m.(\mathtt{out}(\{m\}_B))$

where $B'' = (\textbf{abs}\; x; \mathtt{out}(\{x\}_B \wedge x \neq m))\, B'$.

Some Examples

Communication of Local Names

$$\begin{array}{lcl} P_\pi & = & (\nu b)(\overline{a}b.b(y).R_\pi) \\ Q_\pi & = & a(x).(\nu c)(\overline{x}c) \end{array}$$

$$P_\pi \mid Q_\pi \longrightarrow_\pi^* (\nu b, c) R_\pi[c/y]$$

Some Examples

Communication of Local Names

$$\begin{array}{lcl} P_\pi & = & (\nu b)(\overline{a}b.b(y).R_\pi) \\ Q_\pi & = & a(x).(\nu c)(\overline{x}c) \end{array}$$

$$P_\pi \mid Q_\pi \longrightarrow_\pi^* (\nu b, c) R_\pi[c/y]$$

$$\begin{array}{lcl} P & = & (\textbf{local}\, b)\,(\textbf{tell}(\mathtt{out}(a, b)) \parallel (\textbf{abs}\ y; \mathtt{out}(b, y))\, R) \\ Q & = & (\textbf{abs}\ x; \mathtt{out}(a, x))\,(\textbf{local}\, c)\,(\textbf{tell}(\mathtt{out}(x, c))) \end{array}$$

Some Examples

Communication of Local Names

$$\begin{array}{lcl} P_\pi & = & (\nu b)(\overline{a}b.b(y).R_\pi) \\ Q_\pi & = & a(x).(\nu c)(\overline{x}c) \end{array}$$

$$P_\pi \mid Q_\pi \longrightarrow_\pi^* (\nu b, c) R_\pi[c/y]$$

$$\begin{array}{lcl} P & = & (\textbf{local}\, b)\,(\textbf{tell}(\mathtt{out}(a, b)) \parallel (\textbf{abs}\ y; \mathtt{out}(b, y))\, R) \\ Q & = & (\textbf{abs}\ x; \mathtt{out}(a, x))\,(\textbf{local}\, c)\,(\textbf{tell}(\mathtt{out}(x, c))) \end{array}$$

$$\begin{array}{ll} & P \parallel Q \\ \longrightarrow & (\textbf{local}\, b; \mathtt{out}(a, b))\,((\textbf{abs}\ y; \mathtt{out}(b, y))\, R \parallel \\ & \qquad (\textbf{abs}\ x; \mathtt{out}(a, x))\, Q') \\ \longrightarrow & (\textbf{local}\, b; \mathtt{out}(a, b))\,((\textbf{abs}\ y; \mathtt{out}(b, y))\, R \parallel \\ & \qquad (\textbf{local}\, c)\,(\textbf{tell}(\mathtt{out}(b, c)) \\ \longrightarrow & (\textbf{local}\, b, c; \mathtt{out}(a, b) \wedge \mathtt{out}(b, c))\,((\textbf{abs}\ y; \mathtt{out}(b, y))\, R) \\ \longrightarrow & (\textbf{local}\, b, c; \mathtt{out}(a, b) \wedge \mathtt{out}(b, c))\,(R[c/y]) \end{array}$$

Some Examples

Encoding Recursive Definitions

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

- Proc. definitions: $\ulcorner p(\vec{y}) \stackrel{\texttt{def}}{=} P \urcorner = !\,(\textbf{abs}\ \vec{y}; \mathit{call}_p(\vec{y}))\, \widehat{P}$
  where $\widehat{P}$ is obtained by replacing $p(\vec{x})$ with $\textbf{tell}(\mathit{call}_p(\tilde{x}))$.

Some Examples

Encoding Recursive Definitions

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

- Proc. definitions: $\ulcorner p(\vec{y}) \stackrel{\texttt{def}}{=} P \urcorner = !\,(\textbf{abs}\ \vec{y}; \mathit{call}_p(\vec{y}))\, \widehat{P}$
  where $\widehat{P}$ is obtained by replacing $p(\vec{x})$ with $\textbf{tell}(\mathit{call}_p(\tilde{x}))$.

A simple Example

- Definition:

$$!\,(\textbf{abs}\ N, M, X; \mathit{fact}(N, M, X)) \quad \begin{array}{l} \textbf{when}\ N \leq 1\ \textbf{do}\ \textbf{tell}(X = M) \parallel \\ \textbf{when}\ N > 1\ \textbf{do}\ \textbf{tell}(\mathit{fact}(N - 1, N \times M, X)) \end{array}$$

- Call:

$$\langle \textbf{tell}(\mathit{fact}(3, 1, X)), \texttt{true} \rangle \longrightarrow^* \langle P, X = 6 \rangle$$

Infinite Behavior

No observable transition!!!

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

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

Infinite Behavior

No observable transition!!!

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

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

Example (Message Composition)

$$P \quad = \quad (\textbf{abs}\ x, y; \mathtt{out}(x) \wedge \mathtt{out}(y))\,\textbf{tell}(\mathtt{out}(\{x, y\}))$$

What do we observe from $P \parallel \textbf{tell}(\mathtt{out}(m))$? :
$\mathtt{out}(\{m, m\}), \mathtt{out}(\{m, \{m, m\}\}), \mathtt{out}(\{m, \{m, \{m, m\}\}\}) \ldots$

Outline

Symbolic Semantics for utcc

$$P = \quad (\textbf{abs}\ x, y; \mathtt{out}(x) \wedge \mathtt{out}(y))\, \textbf{tell}(\mathtt{out}(\{x, y\}))$$

Symbolically

$\langle P, \mathtt{out}(m)\rangle \longrightarrow_s$

$\langle \textbf{skip}, \mathtt{out}(m) \wedge \forall x, y : (\mathtt{out}(x) \wedge \mathtt{out}(y) \Rightarrow \mathtt{out}(\{x, y\}))\rangle$

Symbolic Semantics for utcc

$$P = \quad (\textbf{abs}\ x, y; \mathtt{out}(x) \wedge \mathtt{out}(y))\, \textbf{tell}(\mathtt{out}(\{x, y\}))$$

Symbolically

$\langle P, \mathtt{out}(m)\rangle \longrightarrow_s$
$\quad \langle \textbf{skip}, \mathtt{out}(m) \wedge \forall x, y : (\mathtt{out}(x) \wedge \mathtt{out}(y) \Rightarrow \mathtt{out}(\{x, y\}))\rangle$

Temporal Dependencies

$$P = (\textbf{abs}\ x, y; \mathtt{out}(x) \wedge \mathtt{out}(y))\ \textbf{next}\ \textbf{tell}(\mathtt{out}(\{x, y\}))$$

$\langle P, \mathtt{out}(m)\rangle \not\longrightarrow_s$
$\Rightarrow_s \langle \textbf{skip}, \ominus\, \mathtt{out}(m) \wedge \forall x, y : (\ominus(\mathtt{out}(x) \wedge \mathtt{out}(y)) \Rightarrow \mathtt{out}(\{x, y\}))\rangle$

Symbolic Semantics

Symbolic Reductions

$$\mathrm{R}_{\mathrm{ABS-SYM}}\ \frac{\langle P, \exists \vec{x} d\rangle \longrightarrow_s \langle P', \exists \vec{x} d \wedge d'\rangle}{\langle (\mathbf{abs}\ \vec{x}; c)\, P, d\rangle \longrightarrow_s \langle (\mathbf{abs}\ \vec{x}; c)\, P', d \wedge \forall \vec{x}(c \Rightarrow d')\rangle}$$

$$\mathrm{R}_{\mathrm{OBS-SYM}}\ \frac{\langle P, c\rangle \longrightarrow_s^* \langle Q, d\rangle \not\longrightarrow_s}{P \stackrel{(c,d)}{\Longrightarrow} F_s(Q, d)}$$

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

$$F'_s(P) \quad = \quad \begin{cases} (\mathbf{abs}\ \vec{x}; \ominus c)\, F_s(P) & \text{if } P = (\mathbf{abs}\ \vec{x}; c)\, P \\ (\mathbf{local}\ \vec{x}; \ominus c)\, F_s(Q) & \text{if } P = (\mathbf{local}\ \vec{x}; c)\, Q \end{cases}$$

Symbolic Semantics

Symbolic Reductions

$$\mathrm{R}_{\mathrm{ABS-SYM}}\ \frac{\langle P, \exists \vec{x} d\rangle \longrightarrow_s \langle P', \exists \vec{x} d \wedge d'\rangle}{\langle (\mathbf{abs}\ \vec{x}; c)\, P, d\rangle \longrightarrow_s \langle (\mathbf{abs}\ \vec{x}; c)\, P', d \wedge \forall \vec{x}(c \Rightarrow d')\rangle}$$

$$\mathrm{R}_{\mathrm{OBS-SYM}}\ \frac{\langle P, c\rangle \longrightarrow_s^* \langle Q, d\rangle \not\longrightarrow_s}{P \xRightarrow{(c,d)} F_s(Q, d)}$$

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

$$F'_s(P) \quad = \quad \begin{cases} (\mathbf{abs}\ \vec{x}; \ominus c)\, F_s(P) & \text{if } P = (\mathbf{abs}\ \vec{x}; c)\, P \\ (\mathbf{local}\ \vec{x}; \ominus c)\, F_s(Q) & \text{if } P = (\mathbf{local}\ \vec{x}; c)\, Q \end{cases}$$

Theorem (Semantic Correspondence)

Let P be an abstracted-unless free process. The symbolic and the operational outputs of P entail the same basic constraints . ▸ Details

FLTL Correspondence (Declarative view of Processes)

Definition (FLTL Syntax)

$$F, G, \ldots := c \mid F \wedge G \mid \neg F \mid \exists x F \mid \ominus F \mid \circ F \mid \Box F$$

c is a constraint in $\mathcal{L}$. $\Diamond F = \neg\Box\neg F$ (eventually F). ▸ LTL Sem.

FLTL Correspondence (Declarative view of Processes)

Definition (FLTL Syntax)

$$F, G, \ldots := c \mid F \wedge G \mid \neg F \mid \exists x F \mid \ominus F \mid \circ F \mid \Box F$$

c is a constraint in $\mathcal{L}$. $\Diamond F = \neg\Box\neg F$ (eventually F). ▸ LTL Sem.

Definition

FLTL Interpretation of `utccProcesses`

$$\begin{array}{lcllcl}
[\![\textbf{skip}]\!] & = & \texttt{true} & [\![\textbf{tell}(c)]\!] & = & c \\
[\![(\textbf{abs}\ \vec{y}; c)\ P]\!] & = & \forall\vec{y}(c \Rightarrow [\![P]\!]) & [\![P \parallel Q]\!] & = & [\![P]\!] \wedge [\![Q]\!] \\
[\![(\textbf{local}\ \vec{x}; c)\ P]\!] & = & \exists\vec{x}(c \wedge [\![P]\!]) & [\![\textbf{next}\ P]\!] & = & \circ[\![P]\!] \\
[\![\textbf{unless}\ c\ \textbf{next}\ P]\!] & = & c \vee \circ[\![P]\!] & [\![!\ P]\!] & = & \Box[\![P]\!]
\end{array}$$

FLTL Correspondence (Declarative view of Processes)

Definition (FLTL Syntax)

$$F, G, \ldots := c \mid F \wedge G \mid \neg F \mid \exists x F \mid \ominus F \mid \circ F \mid \Box F$$

c is a constraint in $\mathcal{L}$. $\Diamond F = \neg\Box\neg F$ (eventually F). ▸ LTL Sem.

Definition

FLTL Interpretation of `utccProcesses`

$$\begin{array}{lcllcl}
[\![\textbf{skip}]\!] & = & \texttt{true} & [\![\textbf{tell}(c)]\!] & = & c \\
[\![(\textbf{abs}\ \vec{y}; c)\ P]\!] & = & \forall\vec{y}(c \Rightarrow [\![P]\!]) & [\![P \parallel Q]\!] & = & [\![P]\!] \wedge [\![Q]\!] \\
[\![(\textbf{local}\ \vec{x}; c)\ P]\!] & = & \exists\vec{x}(c \wedge [\![P]\!]) & [\![\textbf{next}\ P]\!] & = & \circ[\![P]\!] \\
[\![\textbf{unless}\ c\ \textbf{next}\ P]\!] & = & c \vee \circ[\![P]\!] & [\![!\ P]\!] & = & \Box[\![P]\!]
\end{array}$$

Theorem (Logic Correspondence)

If P is a monotonic process, $[\![P]\!] \models_T \Diamond c$ iff $P \Downarrow_s^c$.

Outline

Undecidability of Monadic FLTL

Theorem (utcc is Turing powerful)

The Minsky machine $M(0,0)$ *halts iff* $[\![M(0,0)]\!] \Downarrow^{\texttt{halt}}$ ▸ Encoding

Undecidability of Monadic FLTL

Theorem (utcc is Turing powerful)

The Minsky machine $M(0,0)$ *halts iff* $[\![M(0,0)]\!] \Downarrow^{\texttt{halt}}$ ▸ Encoding

Monadic FLTL is Incomplete

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

- F is a monadic FLTL formula without functions nor equality.
- Let $G = (F \implies \Box\,\texttt{running})$. G is valid iff M never halts .

Undecidability of Monadic FLTL

Theorem (utcc is Turing powerful)

The Minsky machine $M(0,0)$ *halts iff* $[\![M(0,0)]\!] \Downarrow^{\texttt{halt}}$ ▸ Encoding

Monadic FLTL is Incomplete

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

- F is a monadic FLTL formula without functions nor equality.
- Let $G = (F \implies \Box\,\texttt{running})$. G is valid iff M never halts .

Theorem (Incompleteness of Pnueli's FLTL)

Monadic FLTL without function symbols nor equality is incomplete .

Undecidability of Monadic FLTL

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 drop 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

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. ▸ Details.
- 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 $sp_s(P)$:

$$(w, w') \in io_s(P) \text{ iff } w' = min(sp_s(P) \cap \{w \mid s \leq w\})$$

Denotational Semantics

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

$$\begin{array}{lll}
\text{D}_{\text{TELL}} & [\![\textbf{tell}(c)]\!] & = \{e.w \mid e \models_T c\} \\
\text{D}_{\text{PAR}} & [\![P \parallel Q]\!] & = [\![P]\!] \cap [\![Q]\!] \\
\text{D}_{\text{LOC}} & [\![(\textbf{local}\, \vec{x}; c)\, P]\!] & = \{w \mid \text{there exists an } \vec{x}\text{-variant} \\
 & & \quad w' \text{ of } w \text{ s.t } w'(1) \models_T c \text{ and } w' \in [\![P]\!]\} \\
\text{D}_{\text{ABS}} & [\![(\textbf{abs}\ \vec{x}; c)\, P]\!] & = \{w \mid \text{for every } \vec{x}\text{-variant } w' \text{ of } w \text{ if } w'(1) \models_T c \\
 & & \quad \text{and } w' \succeq (\vec{x} = \vec{t})^{\omega} \text{ for some admissible } \vec{t} \text{ then } w' \in [\![P]\!]\}
\end{array}$$

Denotational Semantics

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

$$
\begin{array}{lll}
\mathrm{D}_{\mathrm{TELL}} & [\![\mathbf{tell}(c)]\!] & = \{e.w \mid e \models_T c\} \\
\mathrm{D}_{\mathrm{PAR}} & [\![P \parallel Q]\!] & = [\![P]\!] \cap [\![Q]\!] \\
\mathrm{D}_{\mathrm{LOC}} & [\![(\mathbf{local}\,\vec{x}; c)\,P]\!] & = \{w \mid \text{there exists an } \vec{x}\text{-variant} \\
 & & \quad w' \text{ of } w \text{ s.t } w'(1) \models_T c \text{ and } w' \in [\![P]\!]\} \\
\mathrm{D}_{\mathrm{ABS}} & [\![(\mathbf{abs}\ \vec{x}; c)\,P]\!] & = \{w \mid \text{for every } \vec{x}\text{-variant } w' \text{ of } w \text{ if } w'(1) \models_T c \\
 & & \quad \text{and } w' \succeq (\vec{x} = \vec{t})^{\omega} \text{ for some admissible } \vec{t} \text{ then } w' \in [\![P]\!]\}
\end{array}
$$

Theorem (Full Abstraction)

Let P, Q be monotonic processes. It holds:

- $P \approx_s^{io} Q$ *iff* $[\![P]\!] = [\![Q]\!]$.

Outline

Closure operator semantics for Sec. Languages.

SCCP Syntax

$$
\begin{array}{llll}
\text{Values} & v, v' & ::= & n \mid x \\
\text{Keys} & k & ::= & pub(v) \mid priv(v) \\
\text{Messages} & M, N & ::= & v \mid k \mid X \mid \{M, N\} \mid \{M\}_k \\
\text{Processes} & R & ::= & \mathbf{nil} \\
 & & & \mid \mathbf{new}(x)R \\
 & & & \mid \mathbf{out}(M).R \\
 & & & \mid \mathbf{in}\ (\vec{x})[M].R \\
 & & & \mid !R \\
 & & & \mid R_i \parallel R_j
\end{array}
$$

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	$::=$	$\mathbf{nil}$	$\Longrightarrow \mathbf{skip}$
			$\mid \mathbf{new}(x)R$	$\Longrightarrow (\mathbf{local}\, x)\, [\![R]\!]$
			$\mid \mathbf{out}(M).R$	$\Longrightarrow ! \, \mathbf{tell}(\mathtt{out}\,(M)) \parallel \mathbf{next}\, [\![R]\!]$
			$\mid \mathbf{in}\ (\vec{x})[M].R$	$\Longrightarrow (\mathbf{abs}\ \vec{x}; \mathtt{out}(M))\, \mathbf{next}\, [\![R]\!]$
			$\mid !R$	$\Longrightarrow ! \, [\![R]\!]$
			$\mid R_i \parallel R_j$	$\Longrightarrow [\![R_i]\!] \parallel [\![R_j]\!]$

Closure operator semantics for Sec. Languages.

SCCP Syntax

$$
\begin{array}{llll}
\text{Values} & v, v' & ::= & n \mid x \\
\text{Keys} & k & ::= & pub(v) \mid priv(v) \\
\text{Messages} & M, N & ::= & v \mid k \mid X \mid \{M, N\} \mid \{M\}_k \\
\text{Processes} & R & ::= & \mathbf{nil} \Longrightarrow \mathbf{skip} \\
 & & & \mid \mathbf{new}(x)R \Longrightarrow (\mathbf{local}\, x)\, [\![R]\!] \\
 & & & \mid \mathbf{out}(M).R \Longrightarrow !\, \mathbf{tell}(\mathtt{out}\,(M)) \| \mathbf{next}\, [\![R]\!] \\
 & & & \mid \mathbf{in}\ (\vec{x})[M].R \Longrightarrow (\mathbf{abs}\ \vec{x}; \mathtt{out}(M))\, \mathbf{next}\, [\![R]\!] \\
 & & & \mid !R \Longrightarrow !\, [\![R]\!] \\
 & & & \mid R_i \parallel R_j \Longrightarrow [\![R_i]\!] \parallel [\![R_j]\!]
\end{array}
$$

Security Constraint System

$$PRJ\ \frac{F \models \mathtt{out}(\{m_1, m_2\})}{F \models \mathtt{out}(m_{1/2})} \qquad DEC\ \frac{F \models \mathtt{out}(k^{-1}) \quad F \models \mathtt{out}(\{m\}_k)}{F \models \mathtt{out}(m)}$$

$$ENC\ \frac{F \models \mathtt{out}(m_1) \quad F \models \mathtt{out}(m_2)}{F \models \mathtt{out}(\{m_1\}_{m_2})} \qquad PAIR\ \frac{F \models \mathtt{out}(m_1) \quad F \models \mathtt{out}(m_2)}{F \models \mathtt{out}(\{m_1, m_2\})}$$

An Example

Denning-Sacco key distribution protocol:

$$\begin{array}{lll} msg_1 & A \rightarrow B & : \quad \{(A, m)\}_{pub(B)} \\ msg_2 & B \rightarrow A & : \quad \{n\}_{pub(m)} \end{array}$$

$$\begin{array}{lcl} Init(A, B) & = & !\,\mathbf{new}(m)\mathbf{out}(\{(m, A)\}_{pub(B)}).\mathbf{nil} \\ Resp(B) & = & !\,\mathbf{in}\ (x, u)[\{(x, u)\}_{priv(B)}]. \\ & & \quad \mathbf{new}(n)(\mathbf{out}(\{n\}_{pub(u)}).\mathbf{nil}) \parallel !\,\mathbf{in}\ [n].\mathbf{out}(attack).\mathbf{nil} \end{array}$$

An Example

Denning-Sacco key distribution protocol:

$$\begin{array}{lll} msg_1 & A \rightarrow B & : \quad \{(A, m)\}_{pub(B)} \\ msg_2 & B \rightarrow A & : \quad \{n\}_{pub(m)} \end{array}$$

$$\begin{array}{rcl} Init(A, B) & = & !\,\mathbf{new}(m)\mathbf{out}(\{(m, A)\}_{pub(B)}).\mathbf{nil} \\ Resp(B) & = & !\,\mathbf{in}\ (x, u)[\{(x, u)\}_{priv(B)}]. \\ & & \quad \mathbf{new}(n)(\mathbf{out}(\{n\}_{pub(u)}).\mathbf{nil}) \parallel !\,\mathbf{in}\ [n].\mathbf{out}(attack).\mathbf{nil} \end{array}$$

Proposition

Let R be a SCCP process, P the utcc process representing R and $f = [\![P]\!] \cap [\![!\,\mathbf{when}\ \mathtt{out}(attack)\ \mathbf{do}\ !\,\mathbf{tell}(\mathtt{false})]\!]$.

- *Symbolic Output : $P \Downarrow_s^{attack}$ iff*
- *Closure Operator Semantics : All the fixed points of f take the form $w.\mathtt{false}^\omega$ iff*
- *FLTL Characterization : $\mathrm{TL}[\![P]\!] \models_T \Diamond attack$.*

Language for Structured Communication (sessions)

Definition (The HVK language (Honda 98))

P,Q	::=	**request** $a(k)$ **in** P	Session Request
	|	**accept** $a(x)$ **in** P	Session Acceptance
	|	$k![\vec{e}]$; P	Data Sending
	|	$k?(x)$ **in** P	Data Reception
	|	$k \lhd l$;P	Label Selection
	|	$k \rhd \{l_1 : P_1 \parallel \cdots \parallel l_n : P_n\}$	Label Branching

Language for Structured Communication (sessions)

Definition (The HVK language (Honda 98))

P,Q	::=	**request** $a(k)$ **in** P	Session Request
	|	**accept** $a(x)$ **in** P	Session Acceptance
	|	$k![\vec{e}]$; P	Data Sending
	|	$k?(x)$ **in** P	Data Reception
	|	$k \lhd l$;P	Label Selection
	|	$k \rhd \{l_1 : P_1 \parallel \cdots \parallel l_n : P_n\}$	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	::=	**request** $a(k)$ **in** P	Session Request
	|	**accept** $a(x)$ **in** P	Session Acceptance
	|	$k![\vec{e}]$; P	Data Sending
	|	$k?(x)$ **in** P	Data Reception
	|	$k \lhd l$;P	Label Selection
	|	$k \rhd \{l_1 : P_1 \parallel \cdots \parallel l_n : P_n\}$	Label Branching

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

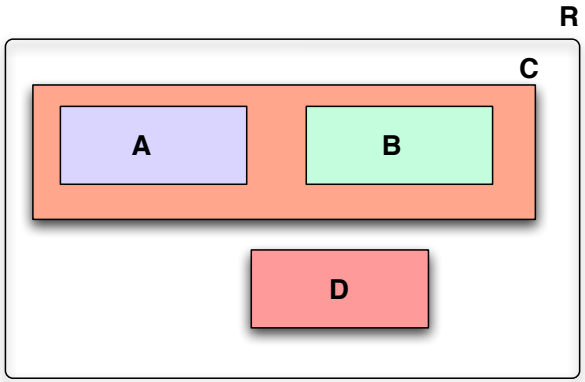
- Sessions should be of finite time.
- One should be able to cancel a session.

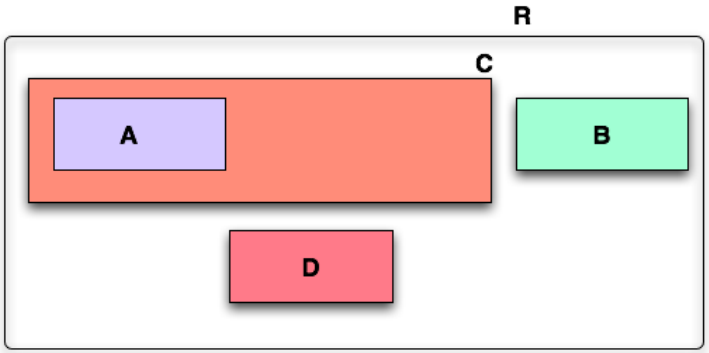
HVK-T

HVK + session cancellation and constraint-guarded **accepts**.

P	::=	**request** $a(k)$ **during** m **in** P	Timed Session Request
	|	**accept** $a(k)$ **given** c **in** P	Declarative Session Acceptance
	|	$\cdots$	
	|	**kill** c_k	Session Abortion

Dynamic Interactive Scores





Dynamic Reconfiguration when Interacting:

- Moving boxes.
- Adding/deleting intervals.

Verification : Minimal conditions to avoid raise conditions.

Outline

Abstract Interpretation of utcc programs

- The behavior of a program P can be approximated by computing $[\![P]\!]^{\tau}_{\alpha}$, a compact representation of $[\![P]\!]$.

Abstract Interpretation of utcc programs

- The behavior of a program P can be approximated by computing $[\![P]\!]^{\tau}_{\alpha}$, a compact representation of $[\![P]\!]$.

Two steps:

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

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

Abstract Interpretation of utcc programs

- The behavior of a program P can be approximated by computing $[\![P]\!]^{\tau}_{\alpha}$, a compact representation of $[\![P]\!]$.

Two steps:

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

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

Abstracting the sequences (τ)

- 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., $\alpha(x = [a|y]) = \mathit{list}(x, y)$
- Secrecy Analysis : Depth-κ cut to approximate the behavior of the protocol:

$$\mathit{cut}_\kappa(m) = \begin{cases} m & \text{if } \mathit{length}(m) \leq \kappa \\ m_\top & \text{otherwise} \end{cases}$$

Outline

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

- C. Olarte and F. Valencia. *Universal Concurrent Constraint Programing: Symbolic Semantics and Applications to Security*. In SAC'08. ACM Press, 2008.
- C. Olarte and F. Valencia. *The Expressivity of Universal Timed CCP: Undecidability of Monadic FLTL and Closure Operators for Security*. In PPDP'08. ACM Press. 2008.
- M. Falaschi, C. Olarte, C. Palamidessi and F. Valencia. *Declarative Diagnosis of Temporal Concurrent Constraint Programs*. In ICLP'07. Springer, 2007.
- M. Falaschi, C. Olarte and C. Palamidessi. *A Framework for Abstract Interpretation of Timed Concurrent Constraint Programs*. In PPDP'09. ACM Press. 2009.
- C. Olarte and C. Rueda. *A declarative language for dynamic multimedia interaction systems*. In MCM'09. Springer, 2009.
- H. A. López, C. Olarte, and J. A. Pérez. *Towards a Unified Framework for Declarative Structured Communications*. In Proc. of PLACES'09.
- C. Olarte and F. Valencia. Undecidability of Monadic First-Order Linear-Time Temporal Logic. Submitted to Studia Logica.

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!


T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard.
Two classes of Boolean functions for dependency analysis.
Science of Computer Programming, 31(1), 1998.


J.A. Bergstra and J.W. Klop.
Algebra of communicating processes with abstraction.
Theoretical Computer Science, 37(1):77–121, 1985.


M. Codish, H. Søndergaard, and P. Stuckey.
Sharing and groundness dependencies in logic programs.
ACM Trans. Program. Lang. Syst., 21(5), 1999.


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.
J. Log. Algebr. Program., 63(1):59–130, 2005.


Francois Fages, Paul Ruet, and Sylvain Soliman.

Linear concurrent constraint programming: Operational and phase semantics.
Information and Computation, 165, 2001.


C. A. R. Hoare.
Communications Sequential Processes.
Prentice-Hall, Englewood Cliffs (NJ), USA, 1985.


Cosimo Laneve and Ugo Montanari.
Mobility in the CC-paradigm.
In *Mathematical Foundations of Computer Science*, 1992.


Stephan Merz.
Decidability and incompleteness results for first-order temporal logics of linear time.
Journal of Applied Non-Classical Logics, 2(2), 1992.


R. Milner.
Communication and Concurrency.
International Series in Computer Science. Prentice Hall, 1989.
SU Fisher Research 511/24.


R. Milner, J. Parrow, and D. Walker.
A calculus of mobile processes, Parts I and II.
Journal of Information and Computation, 100, September 1992.


M. Nielsen, C. Palamidessi, and F.D. Valencia.
Temporal concurrent constraint programming: Denotation, logic and applications.
Nordic Journal of Computing, 9(1), 2002.


Vijay A. Saraswat.
Concurrent Constraint Programming.
MIT Press, 1993.


Vijay Saraswat, Radha Jagadeesan, and Vineet Gupta.
Foundations of timed concurrent constraint programming.
In *Proc. of LICS'94*. IEEE CS, 1994.


Vijay Saraswat and Patrick Lincoln.
Higher-order Linear Concurrent Constraint Programming.
Technical report, 1992.


Frank D. Valencia.
Decidability of infinite-state timed ccp processes and first-order ltl.
Theor. Comput. Sci., 330(3), 2005.

Approaches to mobility in CCP

1. [Sar93] Suppose a injective function f. Let $P = \exists_z \textbf{tell}(x = f(z))$ and $Q = \exists_y(\textbf{ask } x = f(y) \textbf{ then tell}(x = f(y)) \parallel R)$.
   Drawbacks: If two names are sent on x those names must be equals (otherwise an inconsistency arises).
2. [LM92] Let
   $P = \exists z \textbf{tell}(x =< 0, z >)$ and
   $Q = \exists_{y,y'}(\textbf{ask } x =< y, y' > \textbf{ then tell}(x =< y, y' >) \rightarrow R)$.
   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.

Operational Semantics

Constraint System

- A constraint system is a tuple $\langle \sum, \Delta \rangle$ where $\sum$ is a signature and Δ 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 Δ. $c \equiv d$ iff $c \models d$ and $d \models c$
- $\mathcal{C}$ denotes the set of constraints modulo $\equiv$ in $\langle \sum, \Delta \rangle$

Operational Semantics

Structural Congruence

Structural Congruence

Let $\equiv$ be the smallest congruence satisfying:

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

Operational Semantics

Internal Reductions

$$\mathrm{R}_{\mathrm{TELL}}\ \frac{}{\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{skip}, d \wedge c\rangle}$$

$$\mathrm{R}_{\mathrm{PAR}}\ \frac{\langle P, c\rangle \longrightarrow \langle P', d\rangle}{\langle P \parallel Q, c\rangle \longrightarrow \langle P' \parallel Q, d\rangle}$$

$$\mathrm{R}_{\mathrm{LOC}}\ \frac{\langle P, c \wedge (\exists \vec{x} d)\rangle \longrightarrow \langle P', c' \wedge (\exists \vec{x} d)\rangle}{\langle (\mathbf{local}\, \vec{x}; c)\, P, d\rangle \longrightarrow \langle (\mathbf{local}\, \vec{x}; c')\, P', d \wedge \exists \vec{x} c'\rangle}$$

$$\mathrm{R}_{\mathrm{UNL}}\ \frac{d \models_\Delta c}{\langle \mathbf{unless}\ c\ \mathbf{next}\ P, d\rangle \longrightarrow \langle \mathbf{skip}, d\rangle}$$

$$\mathrm{R}_{\mathrm{REP}}\ \frac{}{\langle !\, P, d\rangle \longrightarrow \langle P \parallel \mathbf{next}\, !\, P, d\rangle}$$

$$\mathrm{R}_{\mathrm{ABS}}\ \frac{d \models_\Delta c[\vec{t}/\vec{x}] \qquad [\vec{t}/\vec{x}] \text{ is admissible.}}{\langle (\mathbf{abs}\ \vec{x}; c)\, P, d\rangle \longrightarrow \langle P[\vec{t}/\vec{x}] \parallel (\mathbf{abs}\ \vec{x}; c \wedge \vec{x} \neq \vec{t}\ )\, P, d\rangle}$$

$$\mathrm{R}_{\mathrm{STR}}\ \frac{\gamma_1 \longrightarrow \gamma_2}{\gamma_1' \longrightarrow \gamma_2'}\ \text{if } \gamma_1 \equiv \gamma_1' \text{ and } \gamma_2 \equiv \gamma_2'$$

Operational Semantics

Observable Transition

$$\mathrm{R}_{\mathrm{OBS}} \ \frac{\langle P, c\rangle \longrightarrow^* \langle Q, d\rangle \not\longrightarrow}{P \stackrel{(c,d)}{\Longrightarrow} F(Q)}$$

Future Function

Let F be a partial function defined as:

$$F(P) = \begin{cases} \textbf{skip} & \text{if } P = \textbf{skip} \\ \textbf{skip} & \text{if } P = (\textbf{abs}\ \vec{x}; c)\ Q \\ F(P_1) \parallel F(P_2) & \text{if } P = P_1 \parallel P_2 \\ (\textbf{local}\ \vec{x})\ F(Q) & \text{if } P = (\textbf{local}\ \vec{x}; c)\ Q \\ Q & \text{if } P = \textbf{next}\ Q \\ Q & \text{if } P = \textbf{unless}\ c\ \textbf{next}\ Q \end{cases}$$

Theorem (Semantic Correspondence)

Let P be an abstracted-unless free process. Suppose that $P \xrightarrow{(c_1,d_1)} P_1 \xrightarrow{(c_2,d_2)} \ldots \xrightarrow{(c_i,d_i)} P_i$ *and* $P \xrightarrow{(c_1,e_1)}_s P_1{}' \xrightarrow{(c_2,e_2)}_s \ldots \xrightarrow{(c_i,e_i)} P_i'$. *Then for every $c \in \mathcal{C}$ and $j \in \{1,\ldots,i\}$, $d_i \models c$ iff $e_i \models_T c$.*

Expressiveness of utcc

Minsky Machines

Instructions using two counters c_0 and c_1:

$$
\begin{array}{lcl}
L_i & : & \texttt{HALT} \\
L_i & : & c_n := c_n + 1; \texttt{goto } L_j \\
L_i & : & \texttt{if } c_n = 0 \texttt{ then goto } L_j \texttt{ else } c_n := c_n - 1; \texttt{ goto } L_k
\end{array}
$$

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

$$\texttt{ZERO}_n \stackrel{\text{def}}{=} \textbf{when } \mathit{inc}_n \textbf{ do next } (\textbf{local } a)\,(\texttt{NOT-ZERO}_n(a) \parallel !\,\textbf{when } \texttt{out}(a) \textbf{ do } \texttt{ZERO}_n) \parallel \textbf{when } \mathit{idle}_n \textbf{ do next } \texttt{ZERO}_n \parallel \textbf{tell}(\mathit{isz}_n)$$

$$\texttt{NOT-ZERO}_n(x) \stackrel{\text{def}}{=} \textbf{when } \mathit{inc}_n \textbf{ do next } (\textbf{local } b)\,(\texttt{NOT-ZERO}_n(b) \parallel !\,\textbf{when } \texttt{out}(b) \textbf{ do } \texttt{NOT-ZERO}_n(x)) \parallel \textbf{when } \mathit{dec}_n \textbf{ do next tell}(\texttt{out}(x)) \parallel \textbf{when } \mathit{idle}_n \textbf{ do next } \texttt{NOT-ZERO}_n(x) \parallel \textbf{tell}(\mathit{not\text{-}zero}_n)$$

$$\begin{aligned}
\mathit{ins}(l_i, \texttt{HALT}) &= \textbf{tell}(\texttt{halt}) \parallel \textbf{next tell}(\texttt{out}(l_i)) \parallel \textbf{tell}(\mathit{idle}_0 \wedge \mathit{idle}_1) \\
\mathit{ins}(l_i, \texttt{INC}(c_n, l_j)) &= \textbf{tell}(\mathit{inc}_n) \parallel \textbf{next tell}(\texttt{out}(l_j)) \parallel \textbf{tell}(\mathit{idle}_{1-n}) \\
\mathit{ins}(l_i, \texttt{DECJ}(c_n, l_j, l_k)) &= \textbf{when } \mathit{isz}_n \textbf{ do } (\textbf{next tell}(\texttt{out}(l_j)) \parallel \textbf{tell}(\mathit{idle}_n)) \parallel \\
&\quad \textbf{when } \mathit{not\text{-}zero}_n \textbf{ do } (\textbf{tell}(\mathit{dec}_n) \parallel \textbf{next tell}(\texttt{out}(l_k))) \parallel \\
&\quad \textbf{tell}(\mathit{idle}_{1-n})
\end{aligned}$$

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 $Fix : FF \rightharpoonup \mathcal{P}(FF)$ be defined as

$$
\begin{array}{lcl}
Fix(c) & = & \{F \in FF \mid F \models_T c\} \\
Fix(F_1 \wedge F_2) & = & \{F \in FF \mid F \in Fix(F_1) \text{ and } F \in Fix(F_2)\} \\
Fix(\forall_{\vec{x}} \ominus^n (c) \Rightarrow F_1) & = & \{F \in FF \mid \text{ for all } \vec{x}\text{-variant } F' \text{ of } F, \text{ if } F' \models_T (\ominus^n(c) \wedge \vec{x} = \vec{t}) \\
 & & \quad \text{for some } \vec{t} \in \mathcal{T} \text{ s.t. } adm(\vec{x}, \vec{t}) \text{ then } F' \in Fix(F_1) \} \\
Fix(\exists_{\vec{x}} F_1) & = & \{F \in FF \mid \text{ there exists an } \vec{x}\text{-variant } F' \text{ of } F \text{ s.t. } F' \in Fix(F_1)\} \\
Fix(\ominus F_1) & = & \{F \in FF \mid F = \ominus F' \text{ and } F' \in Fix(F_1)\}
\end{array}
$$

Given the future-free formulae F and G, if $F \in Fix(G)$ we say that F is a fixed formula for G.

Constraint Systems.

A Constraint System defines the basic constraints agents can tell or ask .
A cylindric c.s. is a structure $\mathbf{C} = \langle \mathcal{C}, \leq, \sqcup, \mathtt{true}, \mathtt{false}, \mathit{Var}, \exists, d \rangle$ s.t.:

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

Semantic Equations

$\mathrm{D}_{\mathrm{SKIP}}$	$[\![\textbf{skip}]\!]_I$	$=$	$\mathcal{C}^\omega$
$\mathrm{D}_{\mathrm{TELL}}$	$[\![\textbf{tell}(c)]\!]_I$	$=$	$\{d.s \in \mathcal{C}^\omega \mid d \models c\}$
$\mathrm{D}_{\mathrm{PAR}}$	$[\![P \parallel Q]\!]_I$	$=$	$[\![P]\!]_I \cap [\![Q]\!]_I$
$\mathrm{D}_{\mathrm{NEXT}}$	$[\![\textbf{next}\ P]\!]_I$	$=$	$\{d.s \in \mathcal{C}^\omega \mid s \in [\![P]\!]_I\}$
$\mathrm{D}_{\mathrm{UNL}}$	$[\![\textbf{unless}\ c\ \textbf{next}\ P]\!]_I$	$=$	$\{d.s \in \mathcal{C}^\omega \mid d \not\models c \text{ and } s \in [\![P]\!]_I\}$ $\cup \{d.s \in \mathcal{C}^\omega \mid d \models c\}$
$\mathrm{D}_{\mathrm{REP}}$	$[\![!\ P]\!]_I$	$=$	$\{s \in \mathcal{C}^\omega \mid$ for all w, s' st $s = w.s'$ and $s' \in [\![P]\!]_I\}$
$\mathrm{D}_{\mathrm{LOC}}$	$[\![(\textbf{local}\ \vec{x}; c)\ P]\!]_I$	$=$	$\{s \in \mathcal{C}^\omega \mid$ there exists an $\vec{x}$-variant s' of s st $s'(1) \models c$ and $s' \in [\![P]\!]_I\}$
$\mathrm{D}'_{\mathrm{ABS}}$	$[\![\textbf{when}\ c\ \textbf{do}\ P]\!]_I$	$=$	$\{d.s \in \mathcal{C}^\omega \mid d \models c \text{ and } d.s \in [\![P]\!]_I\}$ $\cup \{d.s \in \mathcal{C}^\omega \mid d \not\models c\}$
$\mathrm{D}_{\mathrm{ABS}}$	$[\![(\textbf{abs}\ \vec{x}; c)\ P]\!]_I$	$=$	$\bigcap_{\vec{t} \in \mathcal{T}^{\lvert\vec{x}\rvert}} [\![(\textbf{when}\ c\ \textbf{do}\ P)[\vec{t}/\vec{x}]]\!]_I$
$\mathrm{D}_{\mathrm{CALL}}$	$[\![p(\vec{x})]\!]_I$	$=$	$I(p(\vec{x}))$

Abs. Semantics Equations

$\mathrm{A}_{\mathrm{SKIP}}$	$[\![\mathbf{skip}]\!]^{\tau}_{X}$	$=$	$\tau(\mathcal{A}^{\omega})$
$\mathrm{A}_{\mathrm{TELL}}$	$[\![\mathbf{tell}(c)]\!]^{\tau}_{X}$	$=$	$\tau(\{d_\kappa.s_\kappa \in \mathcal{A}^{\omega} \mid d_\kappa \models^{\alpha} \alpha(c)\})$
$\mathrm{A}_{\mathrm{PAR}}$	$[\![P \parallel Q]\!]^{\tau}_{X}$	$=$	$[\![P]\!]^{\tau}_{X} \cap [\![Q]\!]^{\tau}_{X}$
$\mathrm{A}_{\mathrm{NEXT}}$	$[\![\mathbf{next}\ P]\!]^{\tau}_{X}$	$=$	$\tau(\{d_\kappa.s_\kappa \in \mathcal{A}^{*} \mid s_\kappa \in [\![P]\!]^{\tau}_{X}\})$
$\mathrm{A}_{\mathrm{UNL}}$	$[\![\mathbf{unless}\ c\ \mathbf{next}\ P]\!]^{\tau}_{X}$	$=$	$\tau(\mathcal{A}^{\omega})$
$\mathrm{A}_{\mathrm{REP}}$	$[\![!\ P]\!]^{\tau}_{X}$	$=$	$\tau(\{s_\kappa \in \mathcal{A}^{*} \mid$ for all s'_κ, w_κ s.t. $s_\kappa = w_\kappa.s'_\kappa, s'_\kappa \in [\![P]\!]^{\tau}_{X}\})$
$\mathrm{A}_{\mathrm{LOC}}$	$[\![(\mathbf{local}\ \vec{x}; c)\ P]\!]^{\tau}_{X}$	$=$	$\tau(\{s_\kappa \in \mathcal{A}^{*} \mid$ there exists a $\vec{x}$-variant s'_κ of s_κ s.t. $s'_\kappa(1) \models^{\alpha} \alpha(c)$ and $s'_\kappa \in [\![P]\!]^{\tau}_{X}\})$
$\mathrm{A}'_{\mathrm{ABS}}$	$[\![\mathbf{when}\ c\ \mathbf{do}\ P]\!]^{\tau}_{X}$	$=$	$\tau(\{d_\kappa.s_\kappa \in \mathcal{A}^{\omega} \mid d_\kappa \not\models_{\mathcal{A}} c\}) \cup \{d_\kappa.s_\kappa \in \mathcal{A}^{*} \mid d_\kappa \models_{\mathcal{A}} c$ and $d_\kappa.s_\kappa \in [\![P]\!]^{\tau}_{X}\}$
$\mathrm{A}_{\mathrm{ABS}}$	$[\![(\mathbf{abs}\ \vec{x}; c)\ P]\!]^{\tau}_{X}$	$=$	$\bigcap_{\vec{t_\kappa} \in \mathcal{T}_\kappa{}^{\lvert\vec{x}\rvert}} [\![(\mathbf{when}\ c\ \mathbf{do}\ P)[\vec{t'}/\vec{x}]]\!]^{\tau}_{X}$ where $\alpha_t(\vec{t'}) = \vec{t_\kappa}$
$\mathrm{A}_{\mathrm{CALL}}$	$[\![p(\vec{x})]\!]^{\tau}_{X}$	$=$	$X(p(\vec{x}))$

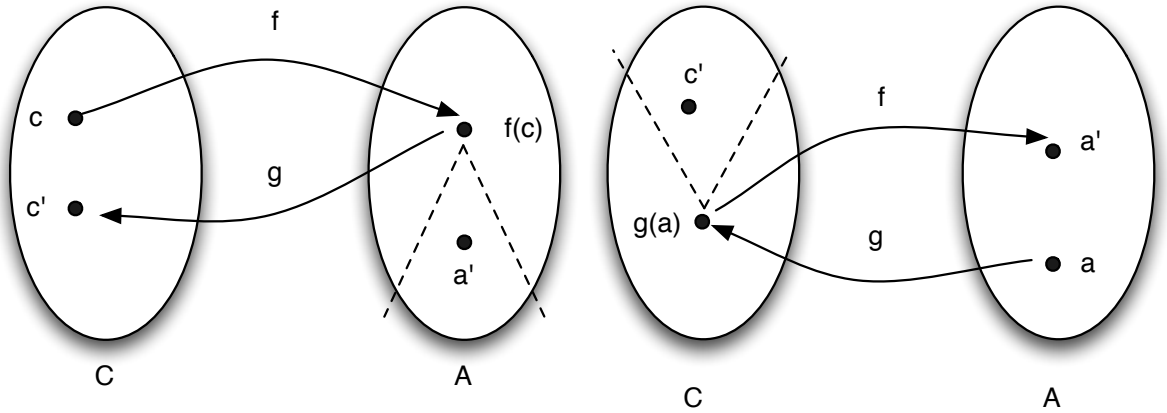
DS Attack

$$
\begin{array}{lcl}
msg_1 & A \rightarrow C & : \quad \{A, m\}_{pub(C)} \\
msg_1' & C \rightarrow B & : \quad \{A, m\}_{pub(B)} \\
msg_2 & B \rightarrow A & : \quad \{n\}_{pub(m)}
\end{array}
$$

Abstract Semantics

Abstract Constraint Systems

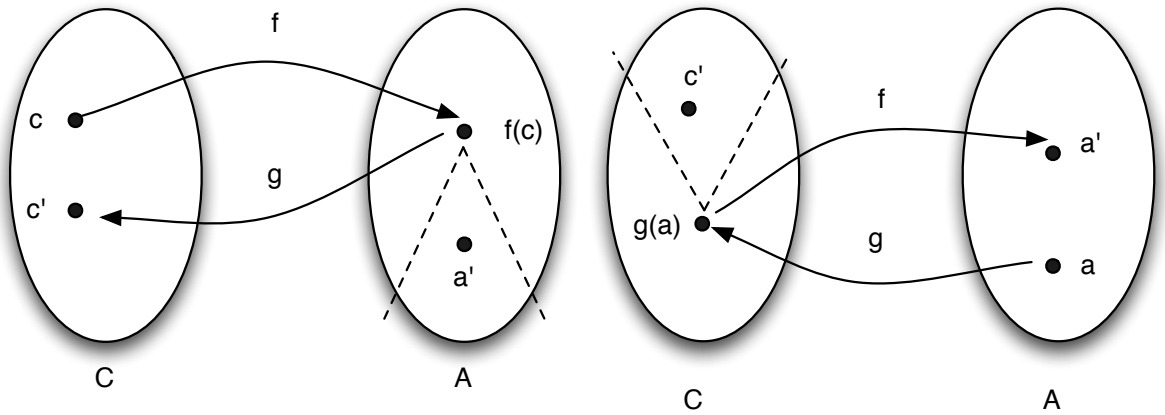
Let **C** and **A** be constraint systems. A description $(\mathcal{C}, \alpha, \mathcal{A})$ consists of an abstract domain $(\mathcal{A}, \leq^{\alpha})$ and a monotonic abstraction function $\alpha : \mathcal{C} \rightarrow \mathcal{A}$.



Abstract Semantics

Abstract Constraint Systems

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



Sequence Abstraction

$\tau : (\mathcal{A}^{\omega} \cup \mathcal{A}^{*}) \rightarrow \mathcal{A}^{*}$ is a reductive operator $(\tau(s_{\kappa}) \leq^{\alpha} s_{\kappa})$.

Abstract Semantics

Semantic Equations

$$
\begin{array}{lll}
\text{A}_{\text{TELL}} & [\![\textbf{tell}(c)]\!]^{\tau}_{X} & = \tau(\{d_\kappa.s_\kappa \in \mathcal{A}^\omega \mid d_\kappa \models^{\alpha} \alpha(c)\}) \\
\text{A}'_{\text{ABS}} & [\![\textbf{when } c \textbf{ do } P]\!]^{\tau}_{X} & = \tau(\{d_\kappa.s_\kappa \in \mathcal{A}^\omega \mid d_\kappa \not\models_{\mathcal{A}} c\}) \\
 & & \cup \{d_\kappa.s_\kappa \in \mathcal{A}^* \mid d_\kappa \models_{\mathcal{A}} c \text{ and } d_\kappa.s_\kappa \in [\![P]\!]^{\tau}_{X}\} \\
\text{A}_{\text{CALL}} & [\![p(\vec{x})]\!]^{\tau}_{X} & = X(p(\vec{x}))
\end{array}
$$

Abstract Semantics

The Abs. semantics is defined as the lfp of

$T^{\alpha}_{\mathcal{D}}(X)(p(\vec{x})) = [\![(\Delta^{\vec{y}}_{\vec{x}} P)]\!]^{\tau}_{X}$ if $p(\vec{y}) \stackrel{\texttt{def}}{=} P \in \mathcal{D}$

Abstract Semantics

Semantic Equations

$$
\begin{array}{lll}
\text{A}_{\text{TELL}} & [\![\textbf{tell}(c)]\!]^{\tau}_{X} & = \tau(\{d_\kappa.s_\kappa \in \mathcal{A}^\omega \mid d_\kappa \models^{\alpha} \alpha(c)\}) \\
\text{A}'_{\text{ABS}} & [\![\textbf{when } c \textbf{ do } P]\!]^{\tau}_{X} & = \tau(\{d_\kappa.s_\kappa \in \mathcal{A}^\omega \mid d_\kappa \not\models_{\mathcal{A}} c\}) \\
 & & \cup \{d_\kappa.s_\kappa \in \mathcal{A}^* \mid d_\kappa \models_{\mathcal{A}} c \text{ and } d_\kappa.s_\kappa \in [\![P]\!]^{\tau}_{X}\} \\
\text{A}_{\text{CALL}} & [\![p(\vec{x})]\!]^{\tau}_{X} & = X(p(\vec{x}))
\end{array}
$$

Abstract Semantics

The Abs. semantics is defined as the lfp of
$T^{\alpha}_{\mathcal{D}}(X)(p(\vec{x})) = [\![(\Delta^{\vec{y}}_{\vec{x}} P)]\!]^{\tau}_{X}$ if $p(\vec{y}) \stackrel{\texttt{def}}{=} P \in \mathcal{D}$

Theorem (Soundness of the approximation)

Given a utcc program $\mathcal{D}.P$, if $s \in [\![P]\!]$ then $\tau(\alpha(s)) \in [\![P]\!]^{\tau}$.

Correctness of the Abstraction

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

Correctness

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

1. $\alpha(\exists_x c) = \exists_x^\alpha \alpha(c)$.
2. $\alpha(d_{xy}) = d_{xy}^\alpha$. And
3. $\alpha(c \sqcup d) \models^\alpha \alpha(c) \sqcup^\alpha \alpha(d)$.

Let α_t be the induced term-abstraction by α. Given the sequence of variables $\vec{x}$ and $\vec{t}, \vec{t'} \in \mathcal{T}^{|\vec{x}|}$, 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 $A = \mathcal{P}(\mathcal{A}^*)$ 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 $A = \mathcal{P}(\mathcal{A}^*)$ and $\subseteq^{\alpha}$ is defined similarly to $\subseteq^{c}$ (Smyth powerdomain). We require $\mathbb{A}$ to be noetherian

Galois Connection

$$\begin{aligned} \alpha(E) &:= \tau(\{\alpha'(s) \mid s \in E\}) \\ \gamma(A) &:= \{s \mid \tau(\alpha'(s)) \in A\} \end{aligned}$$

Soundness of the Abstraction

Abstract domain

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

Galois Connection

$$\begin{aligned} \alpha(E) &:= \tau(\{\alpha'(s) \mid s \in E\}) \\ \gamma(A) &:= \{s \mid \tau(\alpha'(s)) \in A\} \end{aligned}$$

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

$$\begin{aligned} \alpha(I)(p) &:= \tau(\{\alpha'(s) \mid s \in I(p)\}) \\ \gamma(X)(p) &:= \{s \mid \tau(\alpha'(s)) \in X(p)\} \end{aligned}$$