The Expressivity of Universal Timed CCP:
Undecidability of Monadic FLTL and Closure Operators for Security

Carlos Olarte and Frank D. Valencia
INRIA /CNRS and LIX, Ecole Polytechnique
• Concurrent Constraint Prog. (CCP): Agents *telling* and *asking* information represented as *constraints* in a global store.
• Concurrent Constraint Prog. (CCP): Agents *telling* and *asking* information represented as *constraints* in a global store.

• Temporal CCP (tcc): CCP + discrete *time intervals* for modeling *reactive systems*.
• Concurrent Constraint Prog. (CCP): Agents telling and asking information represented as constraints in a global store.

• Temporal CCP (tcc): CCP + discrete time intervals for modeling reactive systems.

  Expressiveness: Processes can be represented as finite-state Büchi automata [Valencia 05]
• Concurrent Constraint Prog. (CCP): Agents telling and asking information represented as constraints in a global store.

• Temporal CCP (tcc): CCP + discrete time intervals for modeling reactive systems.

  ▶ **Expressiveness**: Processes can be represented as finite-state Büchi automata [Valencia 05]

  ▶ **Semantics**: Processes can be compositionally represented as closure operators over sequences of constraints [Saraswat et al 91]
Motivation

• Concurrent Constraint Prog. (CCP): Agents telling and asking information represented as constraints in a global store.

• Temporal CCP (tcc): CCP + discrete time intervals for modeling reactive systems.

  ▶ Expressiveness: Processes can be represented as finite-state Büchi automata [Valencia 05]

  ▶ Semantics: Processes can be compositionally represented as closure operators over sequences of constraints [Saraswat et al 91]

• Universal CCP (utcc): tcc + an abstraction operator for modeling mobile reactive systems. [Olarte & Valencia 08]
Our Contributions

- Expressiveness and Semantic characterization of utcc:
Our Contributions

- Expressiveness and Semantic characterization of utcc:
  1. Proving utcc Turing powerful (Encoding Minsky Machines)
Our Contributions

- Expressiveness and Semantic characterization of utcc:
  1. Proving utcc Turing powerful (Encoding Minsky Machines)
  2. Given a semantic characterization of utcc processes based on closure operators over temporal formulae.
Our Contributions

• Expressiveness and Semantic characterization of utcc:
  1. Proving utcc Turing powerful (Encoding Minsky Machines)
  2. Given a semantic characterization of utcc processes based on closure operators over temporal formulae.

• Applications of the above study:
Our Contributions

• Expressiveness and Semantic characterization of utcc:

1. Proving utcc Turing powerful (Encoding Minsky Machines)

2. Given a semantic characterization of utcc processes based on closure operators over temporal formulae.

• Applications of the above study:

▶ Using (1) to prove that the monadic fragment of first-order linear-time temporal logic (FLTL) is strongly incomplete.
Our Contributions

• Expressiveness and Semantic characterization of utcc:
  1. Proving utcc Turing powerful (Encoding Minsky Machines)
  2. Given a semantic characterization of utcc processes based on closure operators over temporal formulae.

• Applications of the above study:
  ▶ Using (1) to prove that the monadic fragment of first-order linear-time temporal logic (FLTL) is strongly incomplete.
  ▶ By (2) we bring new semantic insights in the modeling of security protocols: A closure op. semantics for a language for security
1. Universal Timed CCP. Description and Operational Semantics
2. Expressiveness of utcc
   2.1. Encoding Minsky Machines into utcc
   2.2. Incompleteness of Monadic FLTL
3. Infinite Behavior and denotational semantics of utcc
   3.1. Symbolic Semantics
   3.2. Closure Operator Semantics for utcc
   3.3. Application: Semantics to Security Languages
4. Concluding Remarks
The CCP Model: Store of partial information. Agents monotonically add (tell) constraints. They synchronize via blocking asks querying the store.
The utcc Calculus

- The CCP Model: Store of partial information. Agents monotonically add (tell) constraints. They synchronize via blocking asks querying the store.

- The TCC Model: Reactive Systems

\[
P \xrightarrow{c_1, d_1} P \xrightarrow{c_2, d_2} P \xrightarrow{c_3, d_3} P
\]
The utcc Calculus

- The CCP Model: Store of partial information. Agents monotonically add (tell) constraints. They synchronize via blocking asks querying the store.

- The TCC Model: Reactive Systems

**Universal tcc:** utcc replaces the tcc ask operator $\text{when } c \text{ do } P$ (executing $P$ if $c$ can be entailed from the store) by a temporary parametric ask $(\text{abs } \vec{x}; c) P$ executing $P[\vec{t}/\vec{x}]$ for each $\vec{t}$ s.t. $c[\vec{t}/\vec{x}]$ can be deduced from the current store.
The utcc Calculus

- The CCP Model: Store of partial information. Agents monotonically add (tell) constraints. They synchronize via blocking asks querying the store.

- The TCC Model: Reactive Systems

Universal tcc: utcc replaces the tcc ask operator \( \text{when } c \text{ do } P \) (executing \( P \) if \( c \) can be entailed from the store) by a temporary parametric ask \( (\text{abs } \bar{x}; c) P \) executing \( P[t/\bar{x}] \) for each \( t \) s.t. \( c[t/\bar{x}] \) can be deduced from the current store.

Syntax: Processes \( P, Q, \ldots \) in utcc are built from constraints in the underlying constraint system by the following syntax:

\[
P, Q := \text{skip} \mid \text{tell}(c) \mid (\text{abs } \bar{x}; c) P \mid P \parallel Q \mid (\text{local } \bar{x}; c) P \mid \text{next } P \mid \text{unless } c \text{ next } P \mid !P
\]
Operational Semantics and FLTL Correspondence

Internal Reductions

\[
\begin{align*}
R_T & \quad \langle \text{tell}(c), d \rangle \rightarrow \langle \text{skip}, d \land c \rangle \\
R_P & \quad \langle P, c \rangle \rightarrow \langle P', d \rangle \\
R_A & \quad d \vdash c[\vec{t}/\vec{x}] \quad |\vec{t}| = |\vec{x}| \\
& \quad \langle (\text{abs } \vec{x}; c) P, d \rangle \rightarrow \langle P[\vec{t}/\vec{x}] \parallel (\text{abs } \vec{x}; c \land \vec{x} \neq \vec{t}) P, d \rangle
\end{align*}
\]

Observable Transition

\[
\begin{align*}
R_O & \quad \langle P, c \rangle \rightarrow^* \langle Q, d \rangle \\
& \quad P \xrightarrow{(c, d)} F(Q)
\end{align*}
\]
Internal Reductions

\[
\begin{align*}
R_T & : \langle \text{tell}(c), d \rangle \implies \langle \text{skip}, d \land c \rangle \\
R_P & : \langle P, c \rangle \implies \langle P', d \rangle \\
R_A & : \langle (\text{abs } \bar{x}; c) P, d \rangle \implies \langle P[\bar{t}/\bar{x}] \parallel (\text{abs } \bar{x}; c \land \bar{x} \neq \bar{t}) P, d \rangle
\end{align*}
\]

Observable Transition

\[
\begin{align*}
R_O & : \langle P, c \rangle \xrightarrow{*} \langle Q, d \rangle \xrightarrow{\rightsquigarrow} \\
& \quad P \xrightarrow{(c,d)} F(Q)
\end{align*}
\]

Notation: Let \( P = P_1 \xrightarrow{(\text{true}, c_1)} P_2 \xrightarrow{(\text{true}, c_2)} \ldots P_i \xrightarrow{(\text{true}, c_i)} \ldots \)

If \( c_i \vdash c \) then we write \( P \Downarrow_c \)
**Operational Semantics and FLTL Correspondence**

### Internal Reductions

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_T$</td>
<td>$\langle \text{tell}(c), d \rangle \rightarrow \langle \text{skip}, d \land c \rangle$</td>
</tr>
<tr>
<td>$R_P$</td>
<td>$\langle P \parallel Q, c \rangle \rightarrow \langle P' \parallel Q, d \rangle$</td>
</tr>
<tr>
<td>$R_A$</td>
<td>$d \vdash c[\bar{t}/\bar{x}] \quad</td>
</tr>
</tbody>
</table>

### Observable Transition

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_O$</td>
<td>$\langle P, c \rangle \rightarrow^* \langle Q, d \rangle \rightarrow$</td>
</tr>
</tbody>
</table>

### Notation:
Let $P = P_1$ $\xrightarrow{(\text{true}, c_1)} P_2$ $\xrightarrow{(\text{true}, c_2)} \ldots P_i$ $\xrightarrow{(\text{true}, c_i)} \ldots$

If $c_i \vdash c$ then we write $P \Downarrow_c$

**Declarative view of utcc Processes based on FLTL formulae**

**Definition:** Let $[\cdot]$ be a map from utcc processes to FLTL formulae:

- $[\text{skip}] = \text{true}$
- $[\text{tell}(c)] = c$
- $[\text{abs} \bar{y}; c] P] = \forall \bar{y}(c \Rightarrow [P])$
- $[\text{local} \bar{x}; c] P] = \exists \bar{x}(c \land [P])$
- $[\text{next} P] = \circ [P]$
- $[\text{unless} c \text{ next} P] = c \lor \circ [P]$
- $[\top] = [\bot]$
- $[P] 

**Theorem:** Logic Correspondence $[P] \vdash_T \diamond c$ iff $P \Downarrow_c$
Agenda

1. Universal Timed CCP. Description and Operational Semantics

2. Expressiveness of utcc
   2.1. Encoding Minsky Machines into utcc
   2.2. Incompleteness of Monadic FLTL

3. Infinite Behavior and denotational semantics of utcc
   3.1. Symbolic Semantics
   3.2. Closure Operator Semantics for utcc
   3.3. Application: Semantics to Security Languages

4. Concluding Remarks
A Minsky Machines is a imperative program consisting of a sequence of labeled instruction modifying the values of two non-negative counters. It computes the value \( n \) if it halts with \( c_0 = n \)

\[
\begin{align*}
L_i : & \text{ halt} \\
L_i : & \text{ } c_n := c_n + 1; \text{ goto } L_j \\
L_i : & \text{ if } c_n = 0 \text{ then goto } L_j \text{ else } c_n := c_n - 1; \text{ goto } L_k
\end{align*}
\]
• A Minsky Machines is a imperative program consisting of a sequence of labeled instruction modifying the values of two non-negative counters. It computes the value \( n \) if it halts with \( c_0 = n \)

\[
\begin{align*}
L_i &: \text{halt} \\
L_i &: c_n := c_n + 1; \text{goto } L_j \\
L_i &: \text{if } c_n = 0 \text{ then goto } L_j \text{ else } c_n := c_n - 1; \text{ goto } L_k
\end{align*}
\]

• Using the monadic fragment of FOL without function symbols nor equality as cs., it is possible to encode Minsky Machines into utcc s.t:

**Theorem. Correctness.** A M. machine \( M(0,0) \) computes the value \( n \) iff

\[
([M(0,0)] \parallel Dec_n) \downarrow_{yes}
\]
Incompleteness of Monadic FLTL

- Proving decidability of monadic FOL requires to obtain prenex forms to reduce to decidability of propositional logic. In FLTL, it is not possible:

Let $F = (x = 42 \land \diamond x \neq 42)$. If $x$ is a flexible variable, $\Box \exists x F$ is satisfiable whereas $\exists x \Box F$ is not. I.e., moving quantifier to the outermost position does not preserve satisfiability.
• Proving decidability of monadic FOL requires to obtain prenex forms to reduce to decidability of propositional logic. In FLTL, it is not possible:

Let $F = (x = 42 \land \circ x \neq 42)$. If $x$ is a flexible variable, $\square \exists x F$ is satisfiable whereas $\exists x \square F$ is not. I.e., moving quantifier to the outermost position does not preserve satisfiability.

• Using the encoding of Minsky Machines into utcc processes and the correspondence between utcc and FLTL, it is possible to prove:
Incompleteness of Monadic FLTL

- Proving decidability of monadic FOL requires to obtain prenex forms to reduce to decidability of propositional logic. In FLTL, it is not possible:

Let $F = (x = 42 \land \diamond x \neq 42)$. If $x$ is a flexible variable, $\Box \exists x F$ is satisfiable whereas $\exists x \Box F$ is not. I.e., moving quantifier to the outermost position does not preserve satisfiability.

- Using the encoding of Minsky Machines into utcc processes and the correspondence between utcc and FLTL, it is possible to prove:

**Proposition.** Given a Minsky machine $M(0,0)$, it is possible to construct a monadic FLTL formula without equality and function symbols $F_M$ s.t $F_M$ is valid iff $M(0,0)$ loops (i.e., it never halts).
• Proving decidability of monadic FOL requires to obtain prenex forms to reduce to decidability of propositional logic. In FLTL, it is not possible:

Let \( F = (x = 42 \land \diamond x \neq 42) \). If \( x \) is a flexible variable, \( \Box \exists x F \) is satisfiable whereas \( \exists x \Box F \) is not. I.e., moving quantifier to the outermost position does not preserve satisfiability.

• Using the encoding of Minsky Machines into utcc processes and the correspondence between utcc and FLTL, it is possible to prove:

**Proposition.** Given a Minsky machine \( M(0,0) \), it is possible to construct a monadic FLTL formula without equality and function symbols \( F_M \) s.t \( F_M \) is valid iff \( M(0,0) \) loops (i.e., it never halts).

**Corollary: Incompleteness.** Monadic FLTL without equality and function symbols is strongly incomplete.
1. Universal Timed CCP. Description and Operational Semantics

2. Expressiveness of utcc
   2.1. Encoding Minsky Machines into utcc
   2.2. Incompleteness of Monadic FLTL

3. Infinite Behavior and denotational semantics of utcc
   3.1. Symbolic Semantics
   3.2. Closure Operator Semantics for utcc
   3.3. Application: Semantics to Security Languages

4. Concluding Remarks
Due to the underlying c.s. and/or loops in abstractions, some utcc procs. are not well-terminated: they can exhibit infinitely many internal reductions.
Due to the underlying c.s. and/or loops in abstractions, some utcc procs. are not well-terminated: they can exhibit infinitely many internal reductions.

Example. Let $P = \text{tell}(y=42)\|(\text{abs } x; x \geq y) \text{next tell}(\text{out}(x))$. Here the SOS never produces an observable transition (infinitely many valid subs. for $x \geq y$).
• Due to the underlying c.s. and/or loops in abstractions, some utcc procs. are not well-terminated: they can exhibit infinitely many internal reductions.

Example. Let \( P = \text{tell}(y=42)\parallel (\text{abs } x; x \geq y) \text{ next tell(out}(x)) \). Here the SOS never produces an observable transition (infinitely many valid subs. for \( x \geq y \))

• The Symbolic Semantics [Olarte & Valencia 08] of utcc aims at representing finitely the infinite behavior of the SOS using temporal formulae:
Due to the underlying c.s. and/or loops in abstractions, some utcc procs. are not well-terminated: they can exhibit infinitely many internal reductions.

Example. Let $P = \text{tell}(y=42) \parallel (\text{abs } x; x \geq y) \text{ next tell(out(x))}$. Here the SOS never produces an observable transition (infinitely many valid subs. for $x \geq y$).

The Symbolic Semantics [Olarte & Valencia 08] of utcc aims at representing finitely the infinite behavior of the SOS using temporal formulae:

Example. Let $P$ as above. The Symbolic semantics outputs the following:

$$\Theta(y = 42) \land \forall x (\Theta(x \geq y) \Rightarrow \text{out}(x))$$
• Due to the underlying c.s. and/or loops in abstractions, some utcc procs. are not well-terminated: they can exhibit infinitely many internal reductions.

**Example.** Let \( P = \text{tell}(y=42) || (\text{abs } x; x \geq y) \text{ next tell}(\text{out}(x)) \). Here the SOS never produces an observable transition (infinitely many valid subs. for \( x \geq y \))

• The **Symbolic Semantics** [Olarte & Valencia 08] of utcc aims at representing finitely the infinite behavior of the SOS using **temporal formulae**:

**Example.** Let \( P \) as above. The Symbolic semantics outputs the following:

\[
\Theta(y = 42) \land \forall x (\Theta(x \geq y) \Rightarrow \text{out}(x))
\]

**Definition. Symbolic IO behavior.** If \( P = P_1 \xrightarrow{(e_1,e'_1)}_s P_2 \xrightarrow{(e_2,e'_2)}_s \ldots \), we write \( P \xrightarrow{(w,w')}_s \) where \( w = e_1.e_2 \ldots \) and \( w' = e'_1.e'_2 \ldots \)

\[
i\sigma_s(P) = \{(w, w') \mid P \xrightarrow{(w,w')}_s \}
\]
Proposition. Closure Properties. The io-behavior of a proc. $P$ is a (partial) closure operator. It satisfies: Extensiveness and Idempotence. If $P$ is monotonic, then $io_s(P)$ also satisfies Monotonicity.
Proposition. Closure Properties. The io-behavior of a proc. $P$ is a (partial) closure operator. It satisfies: Extensiveness and Idempotence. If $P$ is monotonic, then $io_s(P)$ also satisfies Monotonicity.

- Closure Operators are uniquely determined by their set of fixed points:

Definition. Strongest Postcondition.

$$sp_s(P) = \{w \mid (w, w) \in io_s(P)\}$$
Proposition. **Closure Properties.** The io-behavior of a proc. $P$ is a (partial) closure operator. It satisfies: **Extensiveness** and **Idempotence**. If $P$ is monotonic, then $io_s(P)$ also satisfies **Monotonicity**.

- **Closure Operators are uniquely determined by their set of fixed points:**

  **Definition. Strongest Postcondition.**

  $$sp_s(P) = \{w \mid (w, w) \in io_s(P)\}$$

- **The IO-behavior of a monotonic process can be retrieved from its strongest postcondition:**

  **Corollary** Given a monotonic utcc process $P$,

  $$(w, w') \in io_s(P) \iff w' = \min(sp_s(P) \cap \{s' \mid s' \succeq w\})$$
The strongest postcondition can be compositionally characterized:

\[
\begin{align*}
\text{D}_T \quad \llbracket \text{tell}(c) \rrbracket &= \{ e.w \mid e \vdash_T c \} \\
\text{D}_P \quad \llbracket P \parallel Q \rrbracket &= \llbracket P \rrbracket \cap \llbracket Q \rrbracket \\
\text{D}_L \quad \llbracket (\text{local } \vec{x}; c) \rrbracket &= \{ w \mid \text{there exists an } \vec{x}\text{-variant } w' \text{ of } w \text{ s.t. } w'(1) \vdash_T c \text{ and } w' \in \llbracket P \rrbracket \} \\
\text{D}_A \quad \llbracket (\text{abs } \vec{x}; c) \rrbracket &= \{ w \mid \text{for every } \vec{x}\text{-variant } w' \text{ of } w \text{ if } w'(1) \vdash_T c \text{ and } w' \succeq (\vec{x} = \vec{t})^{\omega} \text{ for some } \vec{t} \text{ s.t. } |\vec{x}| = |\vec{t}| \text{ and } \vec{x} \neq \vec{t} \text{ then } w' \in \llbracket P \rrbracket \}\end{align*}
\]
The strongest postcondition can be **compositionally** characterized:

- **Denotational Model**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\text{tell}(c)]</td>
<td>({e.w \mid e \vdash_T c})</td>
</tr>
<tr>
<td>([P \parallel Q])</td>
<td>([P] \cap [Q])</td>
</tr>
<tr>
<td>([\text{local } \vec{x}; c) P])</td>
<td>({w \mid \text{there exists an } \vec{x}\text{-variant } w' \text{ of } w \text{ s.t. } w'(1) \vdash_T c \text{ and } w' \in [P]})</td>
</tr>
<tr>
<td>([\text{abs } \vec{x}; c) P])</td>
<td>({w \mid \text{for every } \vec{x}\text{-variant } w' \text{ of } w \text{ if } w'(1) \vdash_T c \text{ and } w' \succeq (\vec{x} = \vec{t})^\omega \text{ for some } \vec{t} \text{ s.t. }</td>
</tr>
</tbody>
</table>

**Theorem. Full abstraction.** Let \(P\) and \(Q\) be a monotonic processes. Then

\[
P \sim_s^{i_o} Q \iff [P] = [Q].
\]

Thus the denotational semantics allows us to **retrieve compositionally** the IO-behavior of a monotonic process!
- We map a simple language for security into monotonic utcc procs.

**Syntax of SCCP**

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

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

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

Patterns  \( \Pi, \Pi' \) ::= \( v \mid k \mid X \mid (\Pi, \Pi') \)

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

\( |\text{new}(x)R \)  \( \Rightarrow \text{skip} \)

\( |\text{out}(M).R \)  \( \Rightarrow (\text{local } x) I(R) \)

\( |\text{in } [M > \Pi].R \)  \( \Rightarrow !\text{tell(out}(M)) \parallel \text{next } I(R) \)

\( |!R \)  \( \Rightarrow (\text{abs } \bar{x}; c) \text{next } I(R) \)

\( |R \parallel R \)  \( \Rightarrow \parallel_{i \in I} I(R_i) \)
We map a simple language for security into monotonic utcc procs.

Syntax of SCCP

| Values       | $v, v'$ ::= $n \mid x$ |
| Keys         | $k$ ::= $pub(v) \mid priv(v)$ |
| Messages     | $M, N ::= v \mid k \mid X \mid (M, N) \mid \{M\}^k$ |
| Patterns     | $\Pi, \Pi' ::= v \mid k \mid X \mid (\Pi, \Pi')$ |
| Processes    | $R ::= \text{nil}$ |
|              | $|\text{new}(x)R$ |
|              | $|\text{out}(M).R$ |
|              | $|\text{in} [M > \Pi].R$ |
|              | $|! R$ |
|              | $|R \parallel R$ |

$\Rightarrow \text{skip}$

$\Rightarrow (\text{local } x) I(R)$

$\Rightarrow !\text{tell}(\text{out}(M)) \parallel \text{next } I(R)$

$\Rightarrow (\text{abs } \bar{x}; c) \text{next } I(R)$

$\Rightarrow !I(R)$

$\Rightarrow \parallel_{i \in I} I(R_i)$

**Proposition.** Let $R$ be a SCCP process modeling a security protocol.

$$f = [R]_{SCCP} \cap [!\text{when } \text{out}(\text{attack}) \text{ do } !\text{tell} (\text{false})]$$

Therefore, $I(R) \Downarrow_{\text{attack}}$ iff the least-fixed point of $f$ takes the form $w.\text{false}^\omega$
1. Universal Timed CCP. Description and Operational Semantics

2. Expressiveness of utcc
   2.1. Encoding Minsky Machines into utcc
   2.2. Incompleteness of Monadic FLTL

3. Infinite Behavior and denotational semantics of utcc
   3.1. Symbolic Semantics
   3.2. Closure Operator Semantics for utcc
   3.3. Application: Semantics to Security Languages

4. Concluding Remarks
Concluding Remarks

- We study the expressiveness and semantic characterization of utcc processes concluding that:
Concluding Remarks

• We study the expressiveness and semantic characterization of utcc processes concluding that:

  ◦ Well-terminated processes and a monadic constraint system are enough to encode Turing powerful formalisms.
Concluding Remarks

- We study the expressiveness and semantic characterization of utcc processes concluding that:
  - Well-terminated processes and a monadic constraint system are enough to encode Turing powerful formalisms.
  - utcc processes can be represented as (partial) closure operators over sequences of temporal formulae.
Concluding Remarks

• We study the expressiveness and semantic characterization of utcc processes concluding that:
  ▶ Well-terminated processes and a monadic constraint system are enough to encode Turing powerful formalisms.
  ▶ utcc processes can be represented as (partial) closure operators over sequences of temporal formulae.

• We applied the previous results to other fields in comp. science:
Concluding Remarks

- We study the expressiveness and semantic characterization of utcc processes concluding that:
  - Well-terminated processes and a monadic constraint system are enough to encode Turing powerful formalisms.
  - utcc processes can be represented as (partial) closure operators over sequences of temporal formulae.

- We applied the previous results to other fields in comp. science:
  - We proved strongly incomplete the monadic fragment of FLTL without equality and function symbols
Concluding Remarks

• We study the expressiveness and semantic characterization of utcc processes concluding that:
  
  ‣ **Well-terminated processes** and a **monadic constraint system** are enough to encode **Turing powerful formalisms**.
  
  ‣ **utcc processes** can be represented as (partial) **closure operators** over sequences of temporal formulae.

• We applied the previous results to other fields in comp. science:
  
  ‣ We proved **strongly incomplete** the monadic fragment of FLTL without equality and function symbols
  
  ‣ We described a new reasoning technique for the verification of security protocols based on a closure op. semantics for a **language for security**.
Thank You!
**FLTL Syntax and Semantics**

**Definition 7** Given a constraint system with a first-order language $\mathcal{L}$, the LTL formulae we use are given by the syntax:

$$F, G, \ldots := c \mid F \land G \mid \neg F \mid \exists x F \mid \diamond F \mid \circ F \mid \square F.$$  

**Definition 8** We say that $\sigma$ satisfies $F$ in an $\mathcal{L}$-structure $\mathcal{M}(\mathcal{L})$, written $\sigma \models_{\mathcal{M}(\mathcal{L})} F$, iff $\langle \sigma, 0 \rangle \models_{\mathcal{M}(\mathcal{L})} F$ where:

- $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \text{true}$
- $\langle \sigma, i \rangle \not\models_{\mathcal{M}(\mathcal{L})} \text{false}$
- $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} c$ iff $\sigma(i) \models_{\mathcal{M}(\mathcal{L})} c$
- $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \neg F$ iff $\langle \sigma, i \rangle \not\models_{\mathcal{M}(\mathcal{L})} F$
- $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} F \land G$ iff $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} F$ and $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} G$
- $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \bigcirc F$ iff $i > 0$ and $\langle \sigma, i - 1 \rangle \models_{\mathcal{M}(\mathcal{L})} F$
- $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \circ F$ iff $\langle \sigma, i + 1 \rangle \models_{\mathcal{M}(\mathcal{L})} F$
- $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \Box F$ iff for all $j \geq i, \langle \sigma, j \rangle \models_{\mathcal{M}(\mathcal{L})} F$
- $\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \exists x F$ iff for some $x$-variant $\sigma'$ of $\sigma, \langle \sigma', i \rangle \models_{\mathcal{M}(\mathcal{L})} F$

We say that $F$ is valid in $\mathcal{M}(\mathcal{L})$ iff for all $\sigma, \sigma \models_{\mathcal{M}(\mathcal{L})} F$. $F$ is said to be valid if $F$ is valid for every model $\mathcal{M}(\mathcal{L})$. 


Table 1.1: Internal and observable reductions. In $R_A$, $\vec{x} \neq \vec{t}$ denotes $\bigvee_{1 \leq i \leq |\vec{x}|} x_i \neq t_i$. If $|\vec{x}| = 0$, $\vec{x} \neq \vec{t}$ is defined as false.
Symbolic Semantics

\[
\begin{array}{c}
\text{R}_{\text{As}}\quad (P, \exists \bar{x}e) \rightarrow_s (Q, e'' \land \exists \bar{x}e) \\
\text{R}_{\text{Os}}\quad (\text{abs } \bar{x}; e') P, e \rightarrow_s ((\text{abs } \bar{x}; e') Q, e \land \forall \bar{x}(e' \Rightarrow e''))
\end{array}
\]

Table 3.1: Symbolic Rules for Internal and Observable Transitions.

\[
F'(P) = \begin{cases}
\text{skip} & \text{if } P = \text{skip} \\
(\text{abs } \bar{x}; \odot e) F'(Q) & \text{if } P = (\text{abs } \bar{x}; e) Q \\
F'(P_1) \parallel F'(P_2) & \text{if } P = P_1 \parallel P_2 \\
(\text{local } \bar{x}; \odot e) F'(Q) & \text{if } P = (\text{local } \bar{x}; e) Q \\
Q & \text{if } P = \text{next } Q \\
Q & \text{if } P = \text{unless } c \text{ next } Q
\end{cases}
\]

Theorem 4 (Semantic Correspondence [23]) 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 \( P \xrightarrow{c_1, e_1} \ldots \xrightarrow{c_i, e_i} P_i' \). Then for every \( c \in \mathcal{C} \) and \( j \in \{1, \ldots, i\} \), \( d_j \vdash c \) iff \( e_j \vdash_T c \).
Figure 3.1: Denotational Semantics for utcc. The function $\llbracket \cdot \rrbracket$ is of type $\text{Proc} \to \mathcal{P}(\text{PM})$. In $D_A$, $\vec{x} = \vec{t}$ denotes the constraint $\bigwedge_{1 \leq i \leq |\vec{x}|} x_i = t_i$ and $\vec{t} \neq \vec{x}$ stands for point-wise syntactic difference, i.e. $\bigwedge_{1 \leq i \leq |\vec{x}|} t_i \neq x_i$ (see Sect. 1.1). If $|\vec{x}| = 0$ then $\vec{x} = \vec{t}$ and $\vec{x} \neq \vec{t}$ are defined as true.
1 !when \(isz_n\) do
2 \hspace{1em} \textbf{unless} \(inc_n\) \textbf{next} \textbf{tell}(isz_n) \parallel
3 \hspace{1em} \textbf{when} \(inc_n\) \textbf{do} \textbf{next} (local \(a\)) (tell(out^1_n(a)))\parallel
3' \hspace{1em} !\textbf{when} \ out^2_n(a) \do \textbf{tell}(isz_n))
4 \hspace{1em} ||!(abs \(z\); out^1_n(z))
5 \hspace{1em} \textbf{whenever} \(dec_n \lor inc_n\) \textbf{do}
6 \hspace{1em} \textbf{when} \(dec_n\) \textbf{do} \textbf{next} \textbf{tell}(out^2_n(z))\parallel
7 \hspace{1em} \textbf{when} \(inc_n\) \textbf{do} \textbf{next} (local \(b\)) (tell(out^1_n(b)))\parallel
7' \hspace{1em} !\textbf{when} \ out^2_n(b) \do \textbf{tell}(out^1_n(z)))