Concurrency, Time & Constraints

Frank D. Valencia

CNRS, LIX Ecole Polytechnique de Paris

CLEI Cali, Oct 2005
Concurrency

*Concurrent Systems*: Agents (or processes) that interact with each other.

- Systems as networks where arcs represent agent interaction.

- Models for Concurrency: CCS, Pi-Calculus, CSP. Arrows denote Links.
Concurrent, Constraints

In CCP [Saraswat, ’89]: Agents interact via constraints over shared variables.

- Systems as networks where arcs represent agent interaction.

- Arcs as constraints on the (shared-variables of) agents.
Concurrency, Constraints, and Time

As other models, CCP has extended for new and wider phenomena

E.g:

- **Mobility** [Gilbert and Palamidessi ’00, Réty ’98, Rueda & Valencia ’97].

- **Stochastic Behavior** [Saraswat, Jagadeesan ’98, Gupta-Panangaden-Jagadeesan ’99]

**Timed Behavior**

- (Basic) Timed CCP [Gupta-Jagadeesan-Saraswat ’94]
- Timed Default CCP [Gupta-Jagadeesan-Saraswat ’95]
- Hybrid CCP [Gupta-Jagadeesan-Saraswat ’96]
- Timed CCP: the tccp model [DeBoer-Gabbrielli-Meo ’00]
- Nondeterministic (Basic) Timed CCP [Nielsen-Palamidessi-Valencia ’01]
Timed CCP: the tcc model

In the tcc model, as time passes, some constraints may *disappear* or be created.

E.g.: $t = 1$
Timed CCP: the tcc model

In the tcc model, as time passes, some constraints may *disappear* or be created.

E.g.: \[ t = 2 \]
Timed CCP: the tcc model

In the tcc model, as time passes, some constraints may disappear or be created.

E.g.: $t = 3$
Timed CCP: the tcc model

In the tcc model, as time passes, some constraints may **disappear** or be created.

E.g.: $t = 4$
The Goal of this Talk

“One of the outstanding challenges in concurrency is to find the right marriage between logic and behavioural approaches”. R. Milner.

About Timed CCP (I shall argue that):

- It is simple.
- It expresses interesting real-world temporal situations.
- It is rigorously formalized upon process algebra and logic.
- It offers reasoning techniques from denotational semantics and process logic.
Agenda

- Basic Timed CCP intuitions.
- Semantics.
- A Logic and Proof System.
- Applications.
- Behavior.
- Hierarchy of temporal CCP languages
- Future Work
CCP Intuitions: A Typical CCP Scenario

STORE
(MEDIUM)

temperature > 42

temperature = 50?.P

0 < temperature < 100?.Q

temperature < 70
CCP Intuitions: A Typical CCP Scenario

- temperature > 42
- temperature < 70
- 0 < temperature < 100
- temperature = 50
CCP Intuitions: A Typical CCP Scenario

temperature = 50°.P

temperature > 42

temperature < 70

0 < temperature < 100°.Q
CCP Intuitions: A Typical CCP Scenario

- $\text{temperature} > 42$
- $\text{temperature} < 70$
- $\text{temperature} = 50\degree P$
- Q
**CCP Intuitions: A Typical CCP Scenario**

Partial Information (e.g. temperature is some unknown value > 20).
**CCP Intuitions: A Typical CCP Scenario**

- **Partial Information** (e.g. temperature is some *unknown* value > 20).
- **Concurrent Execution** of Processes.
**CCP Intuitions: A Typical CCP Scenario**

- **Partial Information** (e.g. temperature is some *unknown* value > 20).
- **Concurrent Execution** of Processes.
- **Synchronization** via Blocking-Ask.
CCP Intuitions: Representing Partial Information

Definition. A constraint system consists of a signature $\Sigma$ and first-order theory $\Delta$ over $\Sigma$.

- Constraints $a, b, c, \ldots$: formulae over $\Sigma$.

- Relation $\vdash_\Delta$: decidable entailment relation between constraints.

- $\mathcal{C}$: set of constraints under consideration.
Reactive Systems

\[ i_1 \rightarrow P_1 \]

\[ t=1 \]
Reactive Systems

\[ i_1 \rightarrow P_1 \rightarrow O_1 \]

\[ t=1 \]
Reactive Systems

\[
\begin{align*}
&i_1 \quad O_1 \\
&P_1 \quad P_2 \\
&t=1
\end{align*}
\]
Reactive Systems

\[ \begin{align*}
&i_1 \\
&\downarrow \\
&P_1 \quad &O_1 \\
&\uparrow \\
&i_2 \\
&\downarrow \\
&P_2
\end{align*} \]

\[ t=2 \]
Reactive Systems

\[ \begin{align*}
&i_1 & \quad & o_1 & \quad & i_2 & \quad & o_2 \\
& & & & & & & \\
& P_1 & \rightarrow & P_2 \\
& t=2
\end{align*} \]
Reactive Systems

\[ \begin{align*}
&i_1 \quad o_1 \\
\rightarrow \\
&i_2 \quad o_2 \\
\rightarrow \\
&P_1 \rightarrow P_2 \rightarrow P_3 \\
\text{t=3} 
\end{align*} \]
Reactive Systems

• **Stimulus** $i_i$: input information (as a constraint) for $P_i$.

• **Response** $o_i$: output information (as a constraint) of $P_i$.

• **Stimulus-Response** duration: *time interval* (or *time unit*).

**Examples:** PLC’s, RCX Robots, Micro-Controllers, Synchronous Languages.
## Basic tcc Processes

<table>
<thead>
<tr>
<th>Processes</th>
<th>Description</th>
<th>Action within the time interval</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tell(c)</code></td>
<td>telling information</td>
<td>add <code>c</code> to the store</td>
</tr>
</tbody>
</table>
### Basic tcc Processes

<table>
<thead>
<tr>
<th>Processes</th>
<th>Description</th>
<th>Action within the time interval</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tell(c)</code></td>
<td>telling information</td>
<td>add $c$ to the store</td>
</tr>
<tr>
<td><code>when c do P</code></td>
<td>asking information</td>
<td>when $c$ in the store execute $P$</td>
</tr>
</tbody>
</table>
**Basic tcc Processes**

<table>
<thead>
<tr>
<th>Processes</th>
<th>Description</th>
<th>Action within the time interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <code>tell(c)</code></td>
<td>telling information</td>
<td>add <code>c</code> to the store</td>
</tr>
<tr>
<td>• <code>when c do P</code></td>
<td>asking information</td>
<td>when <code>c</code> in the store execute <code>P</code></td>
</tr>
<tr>
<td>• <code>local x in P</code></td>
<td>hiding</td>
<td>execute <code>P</code> with local <code>x</code></td>
</tr>
</tbody>
</table>
# Basic tcc Processes

<table>
<thead>
<tr>
<th>Processes</th>
<th>Description</th>
<th>Action within the time interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <code>tell(c)</code></td>
<td>telling information</td>
<td>add <code>c</code> to the store</td>
</tr>
<tr>
<td>• <code>when c do P</code></td>
<td>asking information</td>
<td>when <code>c</code> in the store execute <code>P</code></td>
</tr>
<tr>
<td>• <code>local x in P</code></td>
<td>hiding</td>
<td>execute <code>P</code> with local <code>x</code></td>
</tr>
<tr>
<td>• <code>next P</code></td>
<td>unit-delay</td>
<td>delay <code>P</code> one time unit</td>
</tr>
</tbody>
</table>
### Basic tcc Processes

<table>
<thead>
<tr>
<th>Processes</th>
<th>Description</th>
<th>Action within the time interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>• tell($c$)</td>
<td>telling information</td>
<td>add $c$ to the store</td>
</tr>
<tr>
<td>• when $c$ do $P$</td>
<td>asking information</td>
<td>when $c$ in the store execute $P$</td>
</tr>
<tr>
<td>• local $x$ in $P$</td>
<td>hiding</td>
<td>execute $P$ with local $x$</td>
</tr>
<tr>
<td>• next $P$</td>
<td>unit-delay</td>
<td>delay $P$ one time unit.</td>
</tr>
<tr>
<td>• unless $c$ next $P$</td>
<td>time-out</td>
<td>unless $c$ now in the store do $P$ next</td>
</tr>
</tbody>
</table>
## Basic tcc Processes

<table>
<thead>
<tr>
<th>Processes</th>
<th>Description</th>
<th>Action within the time interval</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tell(c)</code></td>
<td>telling information</td>
<td>add <code>c</code> to the store</td>
</tr>
<tr>
<td><code>when c do P</code></td>
<td>asking information</td>
<td>when <code>c</code> in the store execute <code>P</code></td>
</tr>
<tr>
<td><code>local x in P</code></td>
<td>hiding</td>
<td>execute <code>P</code> with <code>local x</code></td>
</tr>
<tr>
<td><code>next P</code></td>
<td>unit-delay</td>
<td>delay <code>P</code> one time unit.</td>
</tr>
<tr>
<td><code>unless c next P</code></td>
<td>time-out</td>
<td>unless <code>c</code> now in the store do <code>P</code> next</td>
</tr>
<tr>
<td>`P</td>
<td></td>
<td>Q`</td>
</tr>
</tbody>
</table>
### Basic tcc Processes

<table>
<thead>
<tr>
<th>Processes</th>
<th>Description</th>
<th>Action within the time interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{i \in I} \text{ when } c_i \text{ do } P_i$</td>
<td>guarded choice</td>
<td>choose $P_i$ s.t., $c_i$ in the store</td>
</tr>
</tbody>
</table>
## Basic tcc Processes

<table>
<thead>
<tr>
<th>Processes</th>
<th>Description</th>
<th>Action within the time interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sum_{i \in I} \text{ when } c_i \text{ do } P_i )</td>
<td>guarded choice</td>
<td>choose ( P_i ) s.t., ( c_i ) in the store</td>
</tr>
<tr>
<td>( *P )</td>
<td>unbounded delay</td>
<td>delay ( P ) indefinitely (not forever)</td>
</tr>
</tbody>
</table>
### Basic tcc Processes

<table>
<thead>
<tr>
<th>Processes</th>
<th>Description</th>
<th>Action within the time interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{i \in I} \text{ when } c_i \text{ do } P_i$</td>
<td>guarded choice</td>
<td>choose $P_i$ s.t., $c_i$ in the store</td>
</tr>
<tr>
<td>$*P$</td>
<td>unbounded delay</td>
<td>delay $P$ indefinitely (not forever)</td>
</tr>
<tr>
<td>$!P$</td>
<td>replication</td>
<td>execute $P$ each time unit</td>
</tr>
</tbody>
</table>
Some Derived Constructs

- **Abortion**
  \[
  \text{abort} \overset{\text{def}}{=} !(\text{tell(false)}).
  \]

- **Asynchronous Parallel**
  \[
  P | Q \overset{\text{def}}{=} (\star P \parallel Q) + (P \parallel \star Q)
  \]

- **Bounded Replication**
  \[
  ![t,t']P \overset{\text{def}}{=} \prod_{t \leq i \leq t'} \text{next}^i P
  \]

- **Bounded Delay**
  \[
  \star[t,t']P \overset{\text{def}}{=} \sum_{t \leq i \leq t'} \text{next}^i P
  \]
Power Saver Example

A power saver:

!(unless (lights = off) next ★ tell(lights = off))
Power Saver Example

A refined power saver:

!(unless (lights = off) next ⋆[0,60] tell(lights = off))
Power Saver Example

A more refined one; deterministic power saver:

\[ !(\text{unless } (\text{lights} = \text{off}) \text{ next } \text{tell}(\text{lights} = \text{off})) \]
Operational Semantics

**Internal Transitions:**

\[ RT \quad \langle \text{tell}(c), a \rangle \rightarrow \langle \text{skip}, a \land c \rangle \]

\[ RG \quad \langle \sum_{i \in I} \text{when } c_i \text{ do } P_i, a \rangle \rightarrow \langle P_j, a \rangle \]

\[ RB \quad \langle ! P, a \rangle \rightarrow \langle P \parallel \text{next} ! P, a \rangle \]

\[ RS \quad \langle \ast P, a \rangle \rightarrow \langle \text{next}^n P, a \rangle^{(n \geq 0)} \]

**Observable Transition**

\[ RO \quad \langle P, a \rangle \rightarrow^* \langle Q, a' \rangle \rightarrow \left\{ \begin{array}{l} Q' \quad \text{if } Q = \text{next } Q' \\ Q' \quad \text{if } Q = \text{unless } (c) \text{ next } Q' \\ F(Q_1) \parallel F(Q_2) \quad \text{if } Q = Q_1 \parallel Q_2 \\ \text{local } x \text{ in } F(Q') \quad \text{if } Q = \text{local } x \text{ in } Q' \\ \text{skip} \quad \text{otherwise} \end{array} \right. \]

\[ P \stackrel{(a,a')}{\rightarrow} F(Q) \]
Operational Semantics

- **Internal Transitions:**
  
  \[ RT: \langle \text{tell}(c), a \rangle \rightarrow \langle \text{skip}, a \land c \rangle \]
  \[ RG: \langle \sum_{i \in I} \text{when } c_i \text{ do } P_i, a \rangle \rightarrow \langle P_j, a \rangle \text{ when } a \vdash c_j \]
  \[ RB: \langle \text{!} P, a \rangle \rightarrow \langle P \parallel \text{next} \text{!} P, a \rangle \]
  \[ RS: \langle \star P, a \rangle \rightarrow \langle \text{next}^n P, a \rangle^{(n \geq 0)} \]

- **Observable Transition**

  \[ RO: \langle P, a \rangle \rightarrow^* \langle Q, a' \rangle \rightarrow \langle Q', a' \rangle \]
  \[ P \xrightarrow{(a,a')} F(Q) = \begin{cases} Q' & \text{if } Q = \text{next } Q' \\ Q' & \text{if } Q = \text{unless } (c) \text{ next } Q' \\ F(Q_1) \parallel F(Q_2) & \text{if } Q = Q_1 \parallel Q_2 \\ \text{local } x \text{ in } F(Q') & \text{if } Q = \text{local } x \text{ in } Q' \\ \text{skip} & \text{otherwise} \end{cases} \]
Observations to Make of Processes

**Stimulus-response interaction**

\[ P = P_1 \xrightarrow{(c_1, c'_1)} P_2 \xrightarrow{(c_2, c'_2)} P_3 \xrightarrow{(c_3, c'_3)} \ldots \]

denoted by \( P \xrightarrow{(\alpha, \alpha')} \omega \) with \( \alpha = c_1.c_2 \ldots \) and \( \alpha' = c'_1.c'_2 \ldots \)

**Observable Behavior**

- **Input-Output**  \( io(P) = \{ (\alpha, \alpha') \mid P \xrightarrow{(\alpha, \alpha')} \omega \} \)
- **Output**  \( o(P) = \{ \alpha' \mid P \xrightarrow{\text{true}\omega, \alpha'} \omega \} \)
- **Strongest Postcondition**  \( sp(P) = \{ \alpha' \mid P \xrightarrow{\neg, \alpha'} \omega \} \)
**Strongest-Postcondition Denotational Semantics**

\[
\begin{align*}
\llbracket \text{tell}(a) \rrbracket &= \{ c \cdot \alpha \in C^\omega : c \vdash a, \} \\
\llbracket P \parallel Q \rrbracket &= \llbracket P \rrbracket \cap \llbracket Q \rrbracket \\
\llbracket \text{!}P \rrbracket &= \{ \alpha : \text{for all } \beta \in C^*, \alpha' \in C^\omega : \alpha = \beta.\alpha' \text{ implies } \alpha' \in \llbracket P \rrbracket \} \\
\llbracket \star P \rrbracket &= \{ \beta.\alpha : \beta \in C^*, \alpha \in \llbracket P \rrbracket \} \\
\llbracket \sum_{i \in I} \text{when } (a_i) \text{ do } P_i \rrbracket &= \bigcup_{i \in I} \{ c \cdot \alpha : c \vdash a_i \text{ and } c \cdot \alpha \in \llbracket P_i \rrbracket \} \cup \\
&\quad \bigcap_{i \in I} \{ c \cdot \alpha : c \not\vdash a_i, \alpha \in C^\omega \} \\
\end{align*}
\]

**Definition.**  \( P \) is **locally-independent** iff its guards depend on no local variables.

**Theorem.**  \( sp(P) \subseteq \llbracket P \rrbracket \) and, if \( P \) is a locally-independent, \( sp(P) = \llbracket P \rrbracket \)
Related Work & Road Map

IO Denotation for Timed CCP: [Gupta-Jagadeesan-Saraswat ’94]
Related Work & Road Map

IO Denotation for Basic Timed CCP: [Gupta-Jagadeesan-Saraswat ’94]

IO Denotation for Nondeterministic Timed CCP [DeBoer-Gabrielli-Meo ’01].
Related Work & Road Map

IO Denotation for Basic Timed CCP: [Gupta-Jagedeesan-Saraswat ’94]

IO Denotation Nondeterministic Timed CCP [DeBoer-Gabbrielli-Meo ’01].

SP Denotation for Nondeterministic Basic Timed CCP [Nielsen-Palimidessi-Valencia ’02].
Related Work & Road Map

IO Denotation for Basic Timed CCP: [Gupta-Jagadeesan-Saraswat’94]
IO Denotation Nondeterministic Timed CCP [DeBoer-Gabbrielli-Meo’01].
SP Denotation for Nondeterministic Basic Timed CCP [Nielsen-Palamidessi-Valencia’02].

RoadMap:

• Operational and Denotational Models for Timed CCP
• Coming Next: Logic & Specification.
An LTL Process Logic

**Syntax.** \( A := c \mid A \land A \mid \neg A \mid \exists x A \mid \Diamond A \mid \lozenge A \mid \Box A \)
An LTL Process Logic

Syntax. $A := c | A \land A | \neg A | \exists x A | \Diamond A | \Diamond A | \Box A$

$c$ means "$c$ holds in the current time unit"
An LTL Process Logic

Syntax. $A := c \mid A \land A \mid \neg A \mid \exists x A \mid \diamond A \mid \lozenge A \mid \square A$

- $c$ means "$c$ holds in the current time unit"
- $\square A$ means "$A$ holds always"
An LTL Process Logic

Syntax. \( A := c \mid A \land A \mid \neg A \mid \exists x A \mid \lozenge A \mid \Diamond A \mid \Box A \)

- \( c \) means "c holds in the current time unit"
- \( \Box A \) means "A holds always"
- \( \Diamond A \) means "A eventually holds"
An LTL Process Logic

Syntax. $A := c \mid A \land A \mid \neg A \mid \exists x A \mid \circ A \mid \diamond A \mid \Box A$

- $c$ means “$c$ holds in the current time unit”
- $\Box A$ means “$A$ holds always”
- $\diamond A$ means “$A$ eventually holds”
- $\circ A$ means “$A$ holds in the next time unit”
An LTL Process Logic

**Syntax.** $A := c | A \land A | \neg A | \exists x A | \Diamond A | \Diamond A | \Box A$

- $c$ means "$c$ holds in the current time unit".
- $\Box A$ means "$A$ holds always".
- $\Diamond A$ means "$A$ eventually holds".
- $\Diamond A$ means "$A$ holds in the next time unit"

**Semantics.** Say $\alpha = c_1.c_2.\ldots \models A$ iff $\langle \alpha, 1 \rangle \models A$ where

- $\langle \alpha, i \rangle \models c$ iff $c_i \vdash c$
- $\langle \alpha, i \rangle \models \neg A$ iff $\langle \alpha, i \rangle \not\models A$
- $\langle \alpha, i \rangle \models A_1 \land A_2$ iff $\langle \alpha, i \rangle \models A_1$ and $\langle \alpha, i \rangle \models A_2$
- $\langle \alpha, i \rangle \models \Diamond A$ iff $\langle \alpha, i + 1 \rangle \models A$
- $\langle \alpha, i \rangle \models \Box A$ iff for all $j \geq i \langle \alpha, j \rangle \models A$
- $\langle \alpha, i \rangle \models \Diamond A$ iff there exists $j \geq i$ s.t. $\langle \alpha, j \rangle \models A$
- $\langle \alpha, i \rangle \models \exists x A$ iff there is $\alpha'$ $x$-variant of $\alpha$ s.t. $\langle \alpha', i \rangle \models A$. 
An LTL Process Logic

Syntax. \( A := c \mid A \land A \mid \neg A \mid \exists x A \mid \circ A \mid \diamond A \mid \Box A \)

- \( c \) means “\( c \) holds in the current time unit”.
- \( \Box A \) means “\( A \) holds always”.
- \( \diamond A \) means “\( A \) eventually holds”
- \( \circ A \) means “\( A \) holds in the next time unit”

For example

\[ \text{If } \alpha = (x > 1).(x > 2).(x > 3) \ldots \text{ then } \alpha \models \diamond x > 42 \]
An LTL Process Logic

**Syntax.** $A := c \mid A \land A \mid \neg A \mid \exists x A \mid O A \mid \diamond A \mid \Box A$

- $c$ means “$c$ holds in the current time unit”.
- $\Box A$ means “$A$ holds always”.
- $\diamond A$ means “$A$ eventually holds”
- $O A$ means “$A$ holds in the next time unit”

For example
- If $\alpha = (x > 1) . (x > 2) . (x > 3) . . . . . then \alpha \models \diamond x > 42$

$\Box (A \lor B) \iff \Box A \lor \Box B$ ??
An LTL Process Logic

**Syntax.** \( A := c \mid A \land A \mid \neg A \mid \exists x A \mid \circ A \mid \lozenge A \mid \square A \)

- \( c \) means “\( c \) holds in the current time unit”.
- \( \square A \) means “\( A \) holds always”.
- \( \lozenge A \) means “\( A \) eventually holds”
- \( \circ A \) means “\( A \) holds in the next time unit”

For example
- If \( \alpha = (x > 1).(x > 2).(x > 3) \ldots \) then \( \alpha \models \lozenge x > 42 \)

- \( \square (A \lor B) \iff \square A \lor \square B \)

- \( \circ \lozenge A \iff \lozenge \square A \)
Specification and Satisfaction

Specifications can be expressed as LTL formulae.
Concurrent, Time & Constraints

Specification and Satisfaction

Specifications can be expressed as LTL formulae. We then say:

\[ P \text{ meets } A, \text{ written } P |\!| A, \text{ iff all sequences } P \text{ outputs satisfy } A \]
Specification and Satisfaction

Specifications can be expressed as LTL formulae. We then say:

\[ P \text{ meets } A, \text{ written } P \models A, \text{ iff all sequences } P \text{ outputs satisfy } A \]

E.g.,

\[ !(\text{unless (LightsOff)} \text{ next } \star \text{ tell(LightsOff)}) \models \Diamond (\text{LightsOff}) \]
Specification and Satisfaction

Specifications can be expressed as LTL formulae. We then say:

\[
P \text{ meets } A, \text{ written } P \models A, \text{ iff all sequences } P \text{ outputs satisfy } A
\]

E.g.,
- \(!\text{(unless } (\text{LightsOff}) \text{ next } \star \text{ tell}(\text{LightsOff})) \models \Diamond (\text{LightsOff})\)
- \(!\text{(when } (\text{AlarmGoesOff}) \text{ do tell}(\text{CloseGate})) \models \Box \text{AlarmGoesOff} \Rightarrow \Box \text{CloseGate}\)
Specification and Satisfaction

Specifications can be expressed as LTL formulae. We then say:

\[
P \text{ meets } A, \text{ written } P \models A, \text{ iff all sequences } P \text{ outputs satisfy } A
\]

E.g.,

\[
\bullet \neg(\text{unless (LightsOff) next } \star \text{ tell(LightsOff)}) \models \Diamond(\text{LightsOff})
\]

\[
\bullet \neg(\text{when (AlarmGoesOff) do tell(CloseGate)}) \models \Box \text{AlarmGoesOff } \Rightarrow \Box \text{CloseGate}
\]

But how can we prove \( P \models A \)?
Proof System for $P \vdash A$

\[
\begin{align*}
& \text{tell}(c) \vdash c \ (\text{tell}) \\
& \frac{P \vdash A \quad Q \vdash B}{P \parallel Q \vdash A \land B} \ (\text{par}) \\
& \frac{P \vdash A}{\text{local } x \text{ in } P \vdash \exists x A} \ (\text{hide}) \\
& \frac{P \vdash A}{\text{next } P \vdash \Box A} \ (\text{next}) \\
& \frac{P \vdash A}{!P \vdash \lozenge A} \ (\text{rep}) \\
& \frac{P \vdash A}{\star P \vdash \lozenge A} \ (\text{star}) \\
& \prod_{i \in I} P_i \vdash A_i \ (\text{sum}) \\
& \sum_{i \in I} \text{when } c_i \text{ do } P_i \vdash \bigvee_{i \in I} (c_i \land A_i) \lor \bigwedge_{i \in I} \neg c_i \\
& \frac{P \vdash A \quad A \Rightarrow B}{P \vdash B} \ (\text{rel})
\end{align*}
\]

Theorem. \textbf{(Completeness)} For every locally-independent process $P$,

$P \models A \iff P \vdash A$
Verification \( P \models A \)

Can we prove \( P \models A \) automatically?
Verification $P \models A$

- Can we prove $P \models A$ automatically?

YES, even for infinite-state processes and first-order LTL formulae!
**Verification** \( P \models A \)

- Can we prove \( P \models A \) *automatically*?

YES, even for *infinite-state* processes and *first-order* LTL formulae!

**Theorem.** Given a locally-independent \( P \) and a negation-free \( A \), the problem of whether \( P \models A \) is *decidable*. 
**Verification** \( P \models A \)

- **Can we prove** \( P \models A \) **automatically**?

YES, even for **infinite-state** processes and **first-order** LTL formulae!

**Theorem.** Given a locally-independent \( P \) and a negation-free \( A \), the problem of whether \( P \models A \) is **decidable**.

...and the proof uses the **denotational semantics** rather than the operational semantics !.
Theoretical Applications: Pnueli’s First-Order LTL

Pnueli’s First-Order LTL (FOLTL):

- Syntax like that of the Timed CCP Logic.
Theoretical Applications: Pnueli’s First-Order LTL

Pnueli’s First-Order LTL (FOLTL):

- Syntax like that of the Timed CCP Logic.
- Models are sequences of states
Theoretical Applications: Pnueli’s First-Order LTL

Pnueli’s First-Order LTL (FOLTL):

- Syntax like that of the Timed CCP Logic.
- Models are sequences of states
- Variables can be flexible (i.e., can change as time passes) or rigid.
Theoretical Applications: Pnueli’s First-Order LTL

Pnueli’s First-Order LTL (FOLTL):

- Syntax like that of the Timed CCP Logic.
- Models are sequences of states
- Variables can be flexible (i.e., can change as time passes) or rigid.
- [Abadi ’89] proved the full-language to be undecidable.
Theoretical Applications: Pnueli’s First-Order LTL

Pnueli’s First-Order LTL (FOLTL):

- Syntax like that of the Timed CCP Logic.
- Models are sequences of states
- Variables can be flexible (i.e., can change as time passes) or rigid.
- [Abadi ’89] proved the full-language to be undecidable.
- Several work identifying decidable fragments of FOLTL.
Theoretical Applications: Pnueli’s First-Order LTL

Pnueli’s First-Order LTL (FOLTL):

- Syntax like that of the Timed CCP Logic.
- Models are sequences of states
- Variables can be flexible (i.e., can change as time passes) or rigid.
- [Abadi ’89] proved the full-language to be undecidable.
- Several work identifying decidable fragments of FOLTL.
- Without rigid variables, FOLTL is decidable. Proof by using the theory of Timed CCP.
Programming Applications: Cells

\textbf{Cell} $x : (v)$ models \textit{cell} $x$ \textit{with contents} $v$. 
**Programming Applications: Cells**

- **Cell** $x : (v)$ models cell $x$ with contents $v$.

\[
x : (z) \overset{\text{def}}{=} \text{tell}(x = z) \parallel \text{unless change}(x) \text{ next } x : (z)
\]
Programming Applications: Cells

Cell $x : (v)$ models cell $x$ with contents $v$.

$$x : (z) \overset{\text{def}}{=} \text{tell}(x = z) \parallel \text{unless change}(x) \text{ next } x : (z)$$

Exchange $exch_f(x,y)$ models $y := x ; x := f(x)$. 
Programming Applications: Cells

Cell $x : (v)$ models cell $x$ with contents $v$.

$$x : (z) \overset{\text{def}}{=} \text{tell}(x = z) \parallel \text{unless} \text{change}(x) \text{ next } x : (z)$$

Exchange $\text{exch}_f (x, y)$ models $y := x ; x := f(x)$.

$$\text{exch}_f (x, y) \overset{\text{def}}{=} \sum_v \text{when } (x = v) \text{ do } ( \text{tell}(\text{change}(x)) \parallel \text{tell}(\text{change}(y)) \parallel \text{next}(x : f(v) \parallel y : (v)) )$$
Programming Applications: Cells

**Cell** $x : (v)$ models cell $x$ with contents $v$.

$$x : (z) \overset{\text{def}}{=} \text{tell}(x = z) \parallel \text{unless } \text{change}(x) \text{ next } x : (z)$$

**Exchange** $\text{exch}_f(x, y)$ models $y := x ; x := f(x)$.

$$\text{exch}_f(x, y) \overset{\text{def}}{=} \sum_v \text{when } (x = v) \text{ do } (\text{tell}(\text{change}(x)) \parallel \text{tell}(\text{change}(y)) \parallel \text{next}(x : f(v) \parallel y : (v)))$$

**Example.** $x : (3) \parallel y : (5) \parallel \text{exch}_7(x, y)$
Programming Applications: Cells

- **Cell** $x : (v)$ models cell $x$ with contents $v$.

  $$x : (z) \overset{\text{def}}{=} \text{tell}(x = z) \parallel \text{unless change}(x) \text{ next } x : (z)$$

- **Exchange** $\text{exch}_f(x, y)$ models $y := x ; x := f(x)$.

  $$\text{exch}_f(x, y) \overset{\text{def}}{=} \sum_v \text{when } (x = v) \text{ do ( } \text{tell(change}(x)) \parallel \text{tell(change}(y)) \parallel \text{next}(x : f(v) \parallel y : (v)) \text{ )}$$

**Example.** $x : (3) \parallel y : (5) \parallel \text{exch}_7(x, y) \xrightarrow{\text{.}} x : (7) \parallel y : (3)$. 
Applications: Logic & Proof System at Work

Proposition. \( \text{exch}_f(x, y) \vdash (x = v) \Rightarrow \circ(x = f(v) \land y = v) \).
Applications: Logic & Proof System at Work

**Proposition.** \( \text{exch}_f(x, y) \vdash (x = v) \Rightarrow \Diamond (x = f(v) \land y = v) \).

\[
\begin{align*}
\frac{x : (g(w)) \vdash x = g(w) \quad y : (w) \vdash y = w}{x : (g(w)) \parallel y : (w) \vdash x = g(w) \land y = w} & \text{Pr.1} \quad \text{Pr.1} \\
\frac{\forall w \in \mathcal{D} \quad \text{tell}(\text{change}(x)) \parallel \text{tell}(\text{change}(y)) \parallel \text{next}(x : f(w) \parallel y : (w)) \vdash \Diamond (x = g(w) \land y = w)}{exch_f(x, y) \vdash \bigvee_{w \in \mathcal{D}} (x = w \land \Diamond (x = g(w) \land y = w)) \lor \bigwedge_{w \in \mathcal{D}} \neg x = w} & \text{LPAR} \quad \text{LNEXT} \quad \text{LSUM} \quad \text{LCONS} \\
\frac{exch_f(x, y) \vdash \bigwedge_{w \in \mathcal{D}} (x = w \Rightarrow \Diamond (x = g(w) \land y = w))}{exch_f(x, y) \vdash (x = v \Rightarrow \Diamond (x = g(v) \land y = v))} & \text{LCONS}
\end{align*}
\]
Programming Applications: LEGO Zigzagging

**Specification.** Go *forward* (f), *right* (r) or *left* (l) but DO NOT go:

- f if preceding action was f,
- r if second-to-last action was r, and
- l if second-to-last action was l.
Programming Applications: LEGO Zigzagging

**Specification.** Go forward (f), right (r) or left (l) but DO NOT go:
- f if preceding action was f,
- r if second-to-last action was r, and
- l if second-to-last action was l.

```
GoForward    def =  f_{exch}(act_1, act_2) \parallel \text{tell(} \text{forward})
GoRight      def =  r_{exch}(act_1, act_2) \parallel \text{tell(} \text{right})
GoLeft       def =  l_{exch}(act_1, act_2) \parallel \text{tell(} \text{left})
Zigzag       def =  \begin{cases} 
\text{when}(act_1 \neq f) \text{ do } \text{GoForward} \\
\text{when}(act_2 \neq r) \text{ do } \text{GoRight} \\
\text{when}(act_2 \neq l) \text{ do } \text{GoLeft } \\
\end{cases} \parallel \text{next Zigzag}
StartZigzag  def =  act_1(0) \parallel act_2(0) \parallel \text{Zigzag}
```
Programming Applications: LEGO Zigzagging

**Specification.** Go forward (\(f\)), right (\(r\)) or left (\(l\)) but DO NOT go:

- \(f\) if preceding action was \(f\),
- \(r\) if second-to-last action was \(r\), and
- \(l\) if second-to-last action was \(l\).

\[
\begin{align*}
\text{GoForward} & \overset{\text{def}}{=} f_{\text{exch}}(\text{act}_1,\text{act}_2) \parallel \text{tell}(\text{forward}) \\
\text{GoRight} & \overset{\text{def}}{=} r_{\text{exch}}(\text{act}_1,\text{act}_2) \parallel \text{tell}(\text{right}) \\
\text{GoLeft} & \overset{\text{def}}{=} l_{\text{exch}}(\text{act}_1,\text{act}_2) \parallel \text{tell}(\text{left}) \\
\text{Zigzag} & \overset{\text{def}}{=} ( \begin{align*}
& \text{when}(\text{act}_1 \neq f) \text{ do } \text{GoForward} \\
& \text{when}(\text{act}_2 \neq r) \text{ do } \text{GoRight} \\
& \text{when}(\text{act}_2 \neq l) \text{ do } \text{GoLeft} \\
\end{align*} ) \parallel \text{next Zigzag} \\
\text{StartZigzag} & \overset{\text{def}}{=} \text{act}_1:(0) \parallel \text{act}_2:(0) \parallel \text{Zigzag}
\end{align*}
\]

**Proposition.** \(\text{StartZigzag} \vdash \Box(\Diamond \text{right} \land \Diamond \text{left})\)
A Timed CCP Programming Language for Robots

LMAN (Hurtado&Munoz 2003): A timed ccp reactive programming language for LEGO RCX Robots.
Music Applications: Controlled Improvisation.

Music composition and performance is a complex task of defining and controlling concurrent activity. E.g:

▷ There are $M_1, \ldots, M_m$ musicians (or Voices if you wish). Each $M_i$ is given a three-notes pattern $p_i$ of delays between each note in the block.

▷ Once her block is played, the musician waits for the others to finish their respective blocks before start playing a new one.

▷ The exact time a new block will be started is not specified, but should not be later than $\text{pdur}$; the sum of the durations of all patterns.

▷ Musicians keep playing notes until all of them play a note simultaneously.
Music Applications: Controlled Improvisation

\[ M_i \overset{\text{def}}{=} \sum_{(j,k,l) \in \text{perm}(p_i)} (\text{Play}^i_{(j,k,l)} \ || \ \text{next}^{j+k+l}(\text{flag}_i := 1 \ || \ \text{whenever}(go = 1)\text{ do } \star_{[0,pdur]}M_i)) \]

\[ \text{Play}^i_{(j,k,l)} \overset{\text{def}}{=} ![[0,j-1]tell(note_i = \text{sil})] \ || \ \text{next}^jtell(c_i[\text{note}_i]) \]
\[ \ || \ ![[j+1,j+k-1]tell(note_i = \text{sil})] \ || \ \text{next}^{j+k}tell(c_i[\text{note}_i]) \]
\[ \ || \ ![[j+k+1,j+k+l-1]tell(note_i = \text{sil})] \ || \ \text{next}^{j+k+l}tell(c_i[\text{note}_i]) \]

\[ C \overset{\text{def}}{=} !(\text{when } \bigwedge_{i \in [1,m]}(\text{flag}_i = 1) \land (\text{stop} = 0) \text{ do } \text{tell}(go = 1) \ || \ \prod_{i \in [1,m]}\text{flag}_i := 0) \]
\[ \ || \ \text{next whenever } \bigwedge_{i \in [1,m]}(\text{note}_i \neq \text{sil}) \text{ do stop} := 1 \]
Music Applications: Controlled Improvisation

\[
Init \overset{\text{def}}{=} \prod_{i \in [1, m]} (\text{tell}(c_i[note_i]) \parallel flag_i : 0) \parallel stop : 0
\]

\[
Sys \overset{\text{def}}{=} Init \parallel C \parallel \prod_{i \in [1, m]} M_i
\]

\[\Diamond \text{Notice that } \text{regardless the musicians’ choices} \text{ the system always terminates iff } \]

\[Sys \vdash \Diamond stop = 1.\]

\[\Box \text{Notice that } \text{there are some musicians’ choices} \text{ on which the system terminates iff } \]

\[Sys \not\vdash \Box stop = 0.\]

The above statements can be effectively verified!
More Timed CCP Applications and Languages

- **Music Composition and Performance** (Rueda&Valencia 2004).
- **Biological System** (Olarte&Rueda 2005, Gutierrez&Perez&Rueda 2005).
- **TimedGentzen** (Saraswat 1995): A tcc-based programming language for reactive-systems implemented in Prolog.
- **JCC** (Saraswat&Gupta 2003): An integration of timed ccp into JAVA. See http://www.cse.psu.edu/~saraswat/jcc.html
Related Work & Road Map

Logic & Proof System for Timed CCP: [Gupta-Jagadeesan-Saraswat ’94, ’95]
Related Work & Road Map

Logic & Proof System for Timed CCP: [Gupta-Jagadeesan-Saraswat ’94, ’95]
Logic & Proof System for *Nondeterministic* Timed CCP [DeBoer-Gabrielli-Meo ’01].
Related Work & Road Map

Logic & Proof System for Timed CCP: [Gupta-Jagadeesan-Saraswat ’94,’95]
Logic & Proof System for Nondeterministic Timed CCP [DeBoer-Gabrielli-Meo ’01].
Logic & Proof System Nondeterministic Basic Timed CCP [Nielsen-Palamidessi-Valencia ’02].
Related Work & Road Map

Logic & Proof System for Timed CCP: [Gupta-Jagadeesan-Saraswat ’94,’95]
Logic & Proof System for Nondeterministic Timed CCP [DeBoer-Gabbrielli-Meo ’01].
Logic & Proof System Nondeterministic Basic Timed CCP [Nielsen-Palamidessi-Valencia ’02].
Decidability of Verification [Valencia ’03].
Related Work & Road Map

Logic & Proof System for Timed CCP: [Gupta-Jagadeesan-Saraswat ’94,’95]
Logic & Proof System for Nondeterministic Timed CCP [DeBoer-Gabrielli-Meo ’01].
Logic & Proof System Nondeterministic Basic Timed CCP [Nielsen-Palamidessi-Valencia ’02].
Decidability of Verification [Valencia ’03]
Verification (Model Checking) for Timed CCP [Falaschi and Villanueva ’03]

RoadMap:
- Operational and Denotational Models for Timed CCP
- Timed CCP Logic and its Applications
- Coming Next: Behavioral Equivalences.
Observations to Make of Processes

- **Stimulus-response interaction**

\[ P = P_1 \xrightarrow{(c_1,c'_1)} P_2 \xrightarrow{(c_2,c'_2)} P_3 \xrightarrow{(c_3,c'_3)} \ldots \]

denoted by \( P \xrightarrow{(\alpha,\alpha')} \omega \) with \( \alpha = c_1.c_2 \ldots \) and \( \alpha' = c'_1.c'_2 \ldots \)

**Observable Behavior**

- **Input-Output** \( io(P) = \{(\alpha, \alpha') \mid P \xrightarrow{(\alpha,\alpha')} \omega\} \)

- **Output** \( o(P) = \{\alpha' \mid P \xrightarrow{\text{true} \omega, \alpha'} \omega\} \)

- **Strongest Postcondition** \( sp(P) = \{\alpha' \mid P \xrightarrow{(\omega,\alpha')} \omega\} \)
Behavioral Equivalences

Definition. Let $l \in \{o, io, sp\}$. Define $P \sim_l Q$ iff $l(P) = l(Q)$.

Unfortunately, neither $\sim_{io}$ nor $\sim_o$ are congruences. Let $\approx_{io}$ and $\approx_o$ be the corresponding congruences.

Theorem. $\approx_{io} = \approx_o \subset \approx_{io} \subset \approx_o$. 
Distinguishing Context Characterizations

**Theorem.** Given $P, Q$ and $\sim \in \{\approx_o, \sim_{io}, \sim_{sp}\}$, one can construct a context $C_{(P,Q)}^{(P,Q)}[.]$ such that:

$$P \sim Q \quad \text{if and only if} \quad C_{(P,Q)}^{(P,Q)}[P] \sim_o C_{(P,Q)}^{(P,Q)}[Q]$$

- Interesting consequence of the theorem:

  Decidability of all $\sim_{io}, \sim_{sp}, \approx_o$ and $\approx_{io}$ reduce to that of $\approx_o$.

- Interesting result introduced for the proof:

  Given $P$ one can construct a finite set including all relevant inputs.
Behavioral Equivalence: Decidability.

**Definition.** A star-free $P$ is **locally-deterministic** iff all its summations occur outside of its local processes.

**Theorem.** Given a locally-deterministic $P$ one can effectively construct a Büchi automaton $B_P$ that recognizes $o(P)$.

As a corollary,

**Theorem.** $\approx_o, \approx_{io}, \sim_{io}, \sim_{sp}$ are all decidable for locally-deterministic processes.
Related Work & Road Map

- Decidability of Various Equivalences [Valencia ’03]
Related Work & Road Map

- Decidability of Various Equivalences [Valencia ’03]
- Timed CCP Bisimilarity Equivalence and its Axiomatization [Tini ’00]

RoadMap:

- Operational and Denotational Models for Timed CCP
- Timed CCP Logic and its Applications
- Behavioral Equivalences
- Coming Next: Timed CCP Language Hierarchy.
Variants and their Expressive Power

Basic Timed CCP with the following alternatives for infinite behavior.

- **tcc[Rec]**
  Recursive definitions $A(x_1, \ldots, x_n) \overset{\text{def}}{=} P$ with $fv(P) \subseteq \{x_1, \ldots, x_n\}$.

- **tcc[Rec, Identical Parameters]**
  As above but every call of $A$ in $P$ is of the form $A(x_1, \ldots, x_n)$.

- **tcc[Rec, No Parameters, Dyn. Scoping]**
  Recursive definitions $A \overset{\text{def}}{=} P$ with Dynamic Scoping.

- **tcc[Rec, No Parameters, Static Scoping]**
  Recursive definitions $A \overset{\text{def}}{=} P$ with Static Scoping.
TCC Hierarchy and $\sim_{io}$ (un)decidability.

- Qualitative distinction between dynamic and static scope.
- The results have inspired similar results for CCS.
Final Remarks

Timed CCP combines the declarative view of LTL with the operational-behavioral view from process calculi.
Final Remarks

Timed CCP combines the declarative view of LTL with the operational-behavioral view from process calculi.

“...One of the outstanding challenges in concurrency is to find the right marriage between logic and behavioural approaches”. R. Milner.
Final Remarks

Timed CCP combines the declarative view of LTL with the operational-behavioral view from process calculi.

“One of the outstanding challenges in concurrency is to find the right marriage between logic and behavioural approaches”. R. Milner.

About Timed CCP:

Simple ideas from concurrency and temporal logic.
Final Remarks

Timed CCP combines the declarative view of LTL with the operational-behavioral view from process calculi.

“One of the outstanding challenges in concurrency is to find the right marriage between logic and behavioural approaches”. R. Milner.

About Timed CCP:
- **Simple** ideas from concurrency and temporal logic.
- **It expresses** interesting real-world temporal situations.
Final Remarks

Timed CCP combines the declarative view of LTL with the operational-behavioral view from process calculi.

“...One of the outstanding challenges in concurrency is to find the right marriage between logic and behavioural approaches”. R. Milner.

About Timed CCP:

- Simple ideas from concurrency and temporal logic.
- It expresses interesting real-world temporal situations.
- Formalization upon process algebra and logic.
Final Remarks

Timed CCP combines the declarative view of LTL with the operational-behavioral view from process calculi.

“...One of the outstanding challenges in concurrency is to find the right marriage between logic and behavioural approaches”. R. Milner.

About Timed CCP:

- Simple ideas from concurrency and temporal logic.

- It expresses interesting real-world temporal situations.

- Formalization upon process algebra and logic.

- Techniques from a denotational semantics and process logic.
Ongoing and Future Work

• Implementation of Automatic Tools for analyzing Timed CCP Processes.


• Secure CCP (Ecole Polytechnique, IBM, Univ. Pisa, Javeriana, Univalle).

• Timed CCP for reasoning about Biological Systems (Olarte&Rueda 2005, Gutierrez&Perez 2005).
Examples of Observables

\[
\begin{align*}
\text{when } a \text{ do next} & \quad \text{when } b \text{ do next } \text{tell}(d) \\
+ & \\
\text{when } c \text{ do next } & \quad \text{when } a \text{ do next when } b \text{ do next } \text{tell}(d) \\
\text{tell}(e) & , \\
\end{align*}
\]

Assuming \( a, b, c, d \) and \( e \) mutually exclusive:

- \( o(P) = o(Q) = \{\text{true}^\omega\} \).
- \( io(P) \neq io(Q) \): If \( \alpha = a.c.\text{true}^\omega \) then \( (\alpha, \alpha) \in io(Q) \) but \( (\alpha, \alpha) \notin io(P) \)
- \( sp(P) \neq sp(Q) \): If \( \alpha = a.c.\text{true}^\omega \) then \( \alpha \in sp(Q) \) but \( \alpha \notin sp(P) \).