Concurrency, Time & Constraints

Frank D. Valencia

Uppsala University, SWEDEN
PPS, Paris, Feb., 2004
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. Arcs denote *Links*. 
Concurrency, 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-Palemidessi-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.: $t = 4$
Agenda

- Issues in Concurrency.
- Basic Timed CCP intuitions.
- Semantics.
- A Logic and Proof System.
- Applications.
- Behavior.
- Hierarchy of temporal CCP languages
- Future Work

Which process constructs fit the intended phenomena?
E.g. atomic actions, parallelism, nondeterminism, hiding, recursion, etc.

**Which process constructs fit the intended phenomena?**
E.g. atomic actions, parallelism, nondeterminism, hiding, recursion, etc.

**How should these constructs be endowed with meaning?**
E.g. Operational, Denotational, or algebraic semantics

Which process constructs fit the intended phenomena?
E.g. atomic actions, parallelism, nondeterminism, hiding, recursion, etc.

How should these constructs be endowed with meaning?
E.g. Operational, Denotational, or algebraic semantics.

How should processes be compared?
E.g. Observable Behavior, Process Equivalences, Congruences and their (un)decidability.

Which process constructs fit the intended phenomena?
E.g. atomic actions, parallelism, nondeterminism, hiding, recursion, etc.

How should these constructs be endowed with meaning?
E.g. Operational, Denotational, or algebraic semantics.

How should processes be compared?
E.g. Observable Behavior, Process Equivalences, Congruences and their (un)decidability.

How should process properties be specified and proved?
E.g. Logic for expressing process specifications (like in Hoare's Logic)

How should these constructs be endowed with meaning?
E.g. Operational, Denotational, or algebraic semantics

How should processes be compared?
E.g. Observable Behavior, Process Equivalences, Congruences and their (un)decidability.

How should process properties be specified and proved?
E.g. Logic for expressing process specifications (like in Hoare's Logic)

How expressive the language and its variations are?
E.g. No loss of expressive power for some fragment.
The Goal

“...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.
CCP Intuitions: A Typical CCP Scenario
CCP Intuitions: A Typical CCP Scenario

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

temperature=50\degree P

temperature>42

temperature<70

0<temperature<100\degree Q
CCP Intuitions: A Typical CCP Scenario

- temperature > 42
- temperature < 70
- temperature = 50 °C
**CCP Intuitions: A Typical CCP Scenario**

- **Partial Information** (e.g. temperature is some *unknown* value > 20).
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

$\mathbf{P}_1$

$\mathbf{t}=1$

$\mathbf{i}_1$  $\mathbf{o}_1$
Reactive Systems

$\text{i}_1 \quad \text{o}_1$

$P_1 \rightarrow P_2$

$t=1$
Reactive Systems

\[
\begin{array}{c}
\text{i}_1 \\
\downarrow \\
\text{P}_1 \\
\downarrow \\
\text{P}_2 \\
\end{array}
\]

\[
\begin{array}{c}
\text{O}_1 \\
\downarrow \\
\text{i}_2 \\
\downarrow \\
\text{t}=2
\end{array}
\]
Reactive Systems

\[ i_1 \rightarrow P_1 \rightarrow P_2 \rightarrow o_2 \]

\[ i_2 \rightarrow o_1 \]

\( t=2 \)
Reactive Systems

\[ \begin{array}{c}
\text{i}_1 \rightarrow \text{P}_1 \rightarrow \text{P}_2 \rightarrow \text{P}_3 \\
\text{o}_1 \rightarrow \text{i}_2 \rightarrow \text{o}_2 \\
t=3
\end{array} \]
- **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></td>
<td></td>
<td></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>
</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>
</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>• 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>
</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>\texttt{tell}(c)</td>
<td>telling information</td>
<td>add (c) to the store</td>
</tr>
<tr>
<td>\texttt{when} (c) \texttt{do} (P)</td>
<td>asking information</td>
<td>when (c) in the store execute (P)</td>
</tr>
<tr>
<td>\texttt{local} (x) \texttt{in} (P)</td>
<td>hiding</td>
<td>execute (P) with local (x)</td>
</tr>
<tr>
<td>\texttt{next} (P)</td>
<td>unit-delay</td>
<td>delay (P) one time unit.</td>
</tr>
<tr>
<td>\texttt{unless} (c) \texttt{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>• \texttt{tell(c)}</td>
<td>telling information</td>
<td>add ( c ) to the store</td>
</tr>
<tr>
<td>• \texttt{when \ c \ do \ P}</td>
<td>asking information</td>
<td>when ( c ) in the store execute ( P )</td>
</tr>
<tr>
<td>• \texttt{local \ x \ in \ P}</td>
<td>hiding</td>
<td>execute ( P ) with local ( x )</td>
</tr>
<tr>
<td>• \texttt{next \ P}</td>
<td>unit-delay</td>
<td>delay ( P ) one time unit.</td>
</tr>
<tr>
<td>• \texttt{unless \ c \ next \ P}</td>
<td>time-out</td>
<td>unless ( c ) now in the store do ( P ) next</td>
</tr>
<tr>
<td>( P \parallel Q )</td>
<td>parallelism</td>
<td>execute ( P ) and ( 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>( \star 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} \textbf{when} c_i \textbf{do} P_i ]</td>
<td>guarded choice</td>
<td>choose ( P_i ) s.t. ( c_i ) in the store</td>
</tr>
<tr>
<td>( \star 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>

Temporal CCP

PPS, 2004

UPPSALA UNIVERSITET

39
Some Derived Constructs

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

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

• **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:

!(\textit{unless} (lights = off) \textit{next} \textit{tell}(lights = off))
Operational Semantics

Internal Transitions:

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

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

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

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

Observable Transition

\[
RO \quad \frac{\langle P, a \rangle \rightarrow^* \langle Q, a' \rangle \rightarrow \text{skip}}{P \xrightarrow{(a, a')} \text{F}(Q) =}
\]

\[
\begin{cases} 
Q' & \text{if } Q = \text{next } Q' \\
Q' & \text{if } Q = \text{unless } (c) \text{ next } Q' \\
\text{F}(Q_1) \parallel \text{F}(Q_2) & \text{if } Q = Q_1 \parallel Q_2 \\
\text{local } x \text{ in } \text{F}(Q') & \text{if } Q = \text{local } x \text{ in } Q' \\
\text{skip} & \text{otherwise}
\end{cases}
\]
Operational Semantics

- **Internal Transitions:**

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

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

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

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

**Observable Transition**

\[
RO \quad \langle P, a \rangle \longrightarrow^* \langle Q, a' \rangle \not\Rightarrow \quad \begin{cases} 
Q' & \text{if } Q = \text{next } Q' \\
Q' & \text{if } Q = \text{unless } (c) \text{ next } Q' \\
F(Q_1) \parallel F(Q_2) & \text{if } Q = Q_1 \parallel Q_2 \\
\text{local } x \text{ in } F(Q') & \text{if } Q = \text{local } x \text{ in } Q' \\
\text{skip} & \text{otherwise}
\end{cases}
\]
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**  
  \( \text{i.o}(P) = \{ (\alpha,\alpha') \mid P \xrightarrow{(\alpha,\alpha')} \omega \} \)

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

- **Strongest Postcondition**  
  \( \text{sp}(P) = \{ \alpha' \mid P \xrightarrow{(-\alpha')} \omega \} \)
**Strongest-Postcondition Denotational Semantics**

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

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

**Theorem.** \( sp(P) \subseteq [P] \) and, if *P* is a locally-independent, \( sp(P) = [P] \)
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 Non-deterministic Timed CCP [DeBoer-Gabrielli-Meo ’01].
Related Work & Road Map

IO Denotation for Basic Timed CCP: [Gupta-Jagedeesan-Saraswat '94]
IO Denotation Non-deterministic Timed CCP [DeBoer-Gabrielli-Meo '01].
SP Denotation for Non-deterministic 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. \( A := c \mid A \land A \mid \neg A \mid \exists x A \mid \diamond A \mid \lozenge A \mid \square A \)
An LTL Process Logic

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

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

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

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

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

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

Syntax. $A := c \mid A \land A \mid \neg A \mid \exists_x A \mid \circ A \mid \diamond A \mid \square A$

- $c$ means "$c$ holds in the current time unit"
- $\square A$ 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. $A := c \mid A \land A \mid \neg A \mid \exists_x A \mid O A \mid \Diamond A \mid \Box A$

- $c$ 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”

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 $\langle \alpha, i \rangle \models A_2$
- $\langle \alpha, i \rangle \models O A$ 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. $A := c | A \land A | \neg A | \exists x A | \Diamond A | \Diamond A | \Box A$

- $c$ 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”

For example

- If $\alpha = (x > 1).(x > 2).(x > 3)\ldots$ then $\alpha \models \Diamond x > 42$
An LTL Process Logic

**Syntax.** $A := c \mid A \land A \mid \neg A \mid \exists_x A \mid \Diamond A \mid \square A$

- $c$ means “$c$ holds in the current time unit”.
- $\square A$ means “$A$ holds always”.
- $\Diamond A$ means “$A$ eventually holds”
- $\lozenge A$ means “$A$ holds in the next time unit”

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

$\square (A \lor B) \Leftrightarrow \square A \lor \square B$ ??
An LTL Process Logic

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

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

For example

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

- $\square (A \lor B) \Leftrightarrow \square A \lor \square B$ ??

- $\square \diamond A \Leftrightarrow \diamond \square A$ ??
Specification and Satisfaction

Specifications can be expressed as LTL formulae.
Specification and Satisfaction

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

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

\[\nabla (\text{unless (LightsOff) next } \ast \text{tell(LightsOff))} \models \Diamond (\text{LightsOff})\]
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.,

- \(! (\text{unless (LightsOff)} \text{ next } \star \text{ tell (LightsOff)}) \models \Diamond (\text{LightsOff})\)

- \(\checkmark (! (\text{when (AlarmGoesOff)} \text{ do tell (CloseGate)}) \models \Box \text{AlarmGoesOff} \Rightarrow \Box \text{CloseGate})\)


Specification and Satisfaction

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

$P$ meets $A$, written $P \models A$, iff all sequences $P$ outputs satisfy $A$

E.g.,

- $\Diamond (\text{LightsOff}) \models \Diamond (\text{LightsOff})$

- $\Box \text{AlarmGoesOff} \Rightarrow \Box \text{CloseGate}$

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

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

Theorem. \textbf{(Completeness)} For every $P, A$
\begin{itemize}
\item $P \vdash A$ implies $P \models A$ and
\item $P \models A$ implies $P \vdash A$, if $P$ is locally-independent.
\end{itemize}
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!
Applications: Pnueli’s First-Order LTL

Pnueli’s First-Order LTL (FOLTL):

Syntax like that of the Timed CCP Logic.
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
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.
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.
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.
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 decidably.* Proof by using the theory of Timed CCP.
Applications: Cells

**Cell** \( x : (v) \) models *cell* \( x \) *with contents* \( v \).
Applications: Cells

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

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

\textbf{Cell} \( x : (v) \) models \textit{cell} \( x \) \textit{with contents} \( v \).

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

\textbf{Exchange} \( exch_f(x, y) \) models \( y := x ; x := f(x) \).
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 $y ::= x ; x ::= f(x)$.

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

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

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

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

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

Example. $x : (3) \mid y : (5) \mid \text{exch}_7(x, y)$
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 $y := x ; x := f(x)$.

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

Example. $x : (3) \parallel y : (5) \parallel exch_7(x, y) \overset{\rightarrow}{\longrightarrow} x : (7) \parallel y : (3)$. 
Applications: Logic & Proof System at Work

Proposition. \[ exch_f(x, y) \vdash (x = v) \Rightarrow \circ(x = f(v) \land y = v). \]
Proposition.  \[ \text{exch}_f(x, y) \vdash (x = v) \Rightarrow \diamond (x = f(v) \land y = v) \]
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.
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.

\[
\begin{align*}
\text{GoForward} & \overset{\text{def}}{=} f_{\text{exch}}(act_1, act_2) \parallel \text{tell(}\text{forward}) \\
\text{GoRight} & \overset{\text{def}}{=} r_{\text{exch}}(act_1, act_2) \parallel \text{tell(}\text{right}) \\
\text{GoLeft} & \overset{\text{def}}{=} l_{\text{exch}}(act_1, act_2) \parallel \text{tell(}\text{left}) \\
\text{Zigzag} & \overset{\text{def}}{=} ( \text{when}(act_1 \neq f) \text{ do } \text{GoForward} \\
& + \text{ when}(act_2 \neq r) \text{ do } \text{GoRight} \\
& + \text{ when}(act_2 \neq l) \text{ do } \text{GoLeft} ) \parallel \text{next Zigzag} \\
\text{StartZigzag} & \overset{\text{def}}{=} act_1:(0) \parallel act_2:(0) \parallel \text{Zigzag}
\end{align*}
\]
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.

\[
\begin{align*}
\text{GoForward} & \overset{\text{def}}{=} f_{\text{exch}}(act_1, act_2) \parallel \text{tell}(\text{forward}) \\
\text{GoRight} & \overset{\text{def}}{=} r_{\text{exch}}(act_1, act_2) \parallel \text{tell}(\text{right}) \\
\text{GoLeft} & \overset{\text{def}}{=} l_{\text{exch}}(act_1, act_2) \parallel \text{tell}(\text{left}) \\
\text{Zigzag} & \overset{\text{def}}{=} \left( \begin{array}{c}
\text{when}(act_1 \neq f) \text{ do GoForward} \\
+ \text{when}(act_2 \neq r) \text{ do GoRight} \\
+ \text{when}(act_2 \neq l) \text{ do GoLeft}
\end{array} \right) \parallel \text{next Zigzag} \\
\text{StartZigzag} & \overset{\text{def}}{=} act_1:(0) \parallel act_2:(0) \parallel \text{Zigzag}
\end{align*}
\]

**Proposition.** \( \text{StartZigzag} \models \Box(\Diamond \text{right} \land \Diamond \text{left}) \)
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** \( sp(P) = \{\alpha' \mid P \xrightarrow{(\_\_\_,\alpha')} \omega\} \)
Behavioral Equivalences

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

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

**Theorem.** $\approx_{i_o} = \approx_o \subseteq \sim_{i_o} \subseteq \sim_o$. 
Distinguishing Context Characterizations

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

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

- Interesting consequence of the theorem:

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

- 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. $\approx_o, \approx_{io}, \approx_{io}, \sim_{sp}$ 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 \( f_{v}(P) \subseteq \{x_1, \ldots, x_n\} \).

- **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 involve FSA, PCP, Encodings and Bisimulation.

- The results have inspired similar results for CCS.
Timed CCP Programming Languages


- **JCC** (2003): An integration of timed ccp into the popular JAVA programming language. See http://www.cse.psu.edu/~saraswat/jcc.html

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.
Future Work


- The role of (Timed) CCP in Security.

- Axiomatizations for Timed CCP Equivalences.

- More (un)decidability results for the full calculus and process logic.
Examples of Observables

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

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

- \( o(P) = o(Q) = \{\text{true}\} \).

- \( io(P) \neq io(Q) \): If \( \alpha = a.c.\text{true}^\omega \) then \( (\alpha, \alpha) \in io(Q) \) but \( (\alpha, \alpha) \not\in io(P) \)

- \( sp(P) \neq sp(Q) \): If \( \alpha = a.c.\text{true}^\omega \) then \( \alpha \in sp(Q) \) but \( \alpha \not\in sp(P) \).