Temporal Concurrent Constraint Programming

Frank D. Valencia's Ph.D Defense

Supervisor: Mogens Nielsen

BRICS, University of Aarhus
Aarhus, Denmark
February, 2003

BRICS

Motivation: Concurrency and Time

Concurrent Systems: Multiple **agents** (**processes**) that interact among each other (e.g., Internet).

- Synchronous Systems:
  Agents need to synchronize (e.g., phone calls).

- Mobile Systems:
  Agents can change their communication links (e.g., *mobile phones*).

- **Timed Systems**:
  Agents are constrained by timed requirements. (e.g., people at a company, music performance, micro-controllers).

Motivation: Our Goal

A CCP model for **describing and analyzing** timed systems.

- **Timed Systems** involve:
  - ▷ **constraints specifying** concurrent behavior
  - ▷ **partial information**
  - ▷ **specific domains** applications

- **CCP** used for:
  - ▷ **specifying** concurrency via **constraints**
  - ▷ manipulating **partial information**
  - ▷ defining **domain specific** programming languages.

The model we developed is the **ntcc calculus**; the subject of this talk.

Other models: Guidelines

- **Models confine themselves to specific computational phenomena**

  E.g. The π calculus: *mobility* and *synchronous* communication.

- **Models must be simple, expressive, formal and provide techniques.**

  E.g. The π calculus:

  ▷ simple idea of naming,

  ▷ full expressive power,

  ▷ formal process theory,

  ▷ bisimulation techniques.

- **Modern models often arise as generalizations (or extensions) of mature models.**

  E.g., The π calculus: a generalization of CCS.

Our Model: The ntcc calculus.

- **ntcc confines itself to timed systems where**:
  computation evolves in *discrete time intervals* (*or time units*).

- **ntcc is simple, expressive, formal and provide techniques.**
  - ▷ 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.

- **ntcc arises as a generalization of tcc (Saraswat et al, 94).**
  Extends the computational model of tcc to allow for nondeterminism and asynchrony.

Models for Concurrency: Key Issues Addressed.

- **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 (Hennessy-Milner Logic)

- **How expressive are the constructs ?**
  E.g., for the π calculus: Asynchronous vs. synchronous version.

Our Model: Key Issues Addressed.

- **Which process constructs fit discrete-timed systems?**
  Nondeterminism, replication, unbounded delays, unit delays and time-outs.

- **How are these constructs given meaning ?**
  Operational and Denotational Semantics.

- **How are ntcc processes compared ?**
  Behavioral equivalences, associated congruences and their (un)decidability.

- **How are ntcc process properties specified and proved ?**
  E.g, Process Logic and associated proof system.

- **How expressive are the ntcc constructs ?**
  E.g., Expressive power hierarchy of variations of ntcc.

Our Model: The main benefit.

ntcc combines the declarative view of *temporal logic* with the operational-behavioral view from *process calculi* . Thus, it benefits from two well-established approaches in concurrency theory.

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

General Contributions

1. A simple yet expressive **model** for timed systems.

2. Extending the **operational & temporal logic** interpretation of processes.

3. Adapting to CCP **techniques** used in **concurrency theory**

4. Using ntcc theory to study **pre-existing** CCP Languages:
   - *First Temp. CCP expressive-power hierarchy*
   - *(Un)Decidability results for their equivalences.*

That is, our work **extends** and **strengthens** the CCP theory of concurrency.

The Rest of this Talk: Overview of our work

Agenda

▷ CCP Intuitions.

▷ Ntcc intuitions.

▷ Operational Semantics.

▷ Denotational Semantics.

▷ Logic and Proof System.

▷ Applications.

▷ Behavioral equivalences, congruences and their decidability.

▷ Hierarchy of temporal CCP languages and (un)decidability of their equivalences.

CCP Intuitions: A Typical CCP Scenario

$(\texttt{temperature} > 20)!$ ↘ **M E D I U M** (Store) ↗ $(\texttt{temperature} = 30)?.P$

$(\texttt{temperature} < 40)!$ ↗ **M E D I U M** (Store) ↘ $(\texttt{0<temperature} < 44)?.Q$

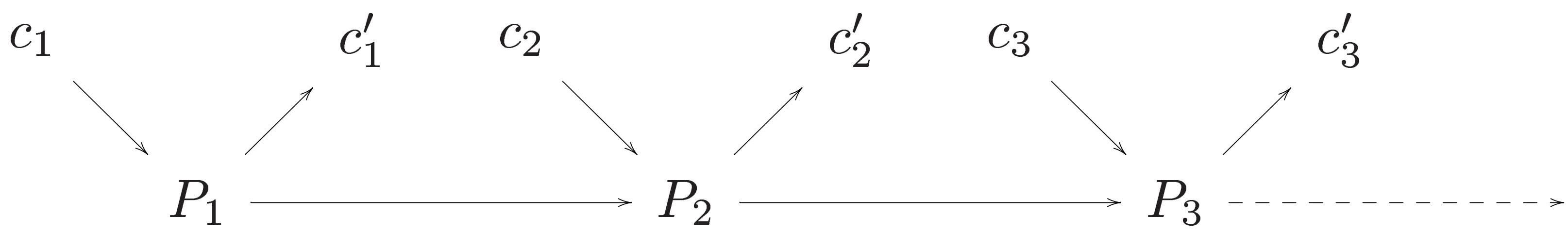
- **Partial Information** (e.g. `temperature` is some *unknown* value > 20).
- **Concurrent Execution** of Processes.
- **Synchronization** via Blocking-Ask.

CCP Intuitions: Representing Partial Information

Definition. *A* **constraint system** *consists of a signature* Σ *and first-order theory* Δ *over* Σ.

- **Constraints** $a, b, c, ...$: formulae over Σ.
- **Relation** $\vdash_\Delta$: decidable entailment relation between constraints.
- $\mathcal{C}$: set of constraints under consideration.

Ntcc Intuitions: Systems that Concern us



- **Stimulus** c_i : input information for P_i .
- **Response** c'_i: output information of P_i .
- **Stimulus-Response** duration: **time interval** (or **time unit**).

Examples: Programmable Logic Controllers (PLC's), LEGO RCX bricks and micro-controllers in general.

Ntcc Syntax : Basic tcc Processes

Processes	Description	Action within the time interval
• $\mathbf{tell}(c)$	telling information	add c to the store
• $\mathbf{when}\ c\ \mathbf{do}\ P$	asking information	when c in the store execute P
• $\mathbf{local}\ x\ \mathbf{in}\ P$	hiding	execute P with local x
• $\mathbf{next}\ P$	unit-delay	delay P one time unit.
• $\mathbf{unless}\ c\ \mathbf{next}\ P$	time-out	unless c now in the store do P next
• $P \parallel Q$	parallelism	execute P and Q

Ntcc Additional Basic Processes

- **Non Deterministic Behavior:** $\sum_{i \in I} \mathbf{when}\, c_i\, \mathbf{do}\, P_i$

  Guarded Choice.

- **Asynchronous Behavior:** $\star P$

  Unbounded but finite delay of P

- **Infinite Behavior**: $!P$

  Unboundely many copies of P, one at a time: $P \parallel \mathbf{next}\, P \parallel \mathbf{next}^2 P \parallel \ldots$

Some Derived Constructs

- **Inactivity:** $\mathbf{skip} \stackrel{\text{def}}{=} \sum_{i \in \emptyset} P_i$

  $P \parallel \mathbf{skip} = P$.

- **Abortion**: $\mathbf{abort} \stackrel{\text{def}}{=} !\,(\mathbf{tell}(\mathtt{false}))$

  $P \parallel \mathbf{abort} = \mathbf{abort}$.

- **Fair asynchronous parallel**: $P \,|\, Q \stackrel{\text{def}}{=} (\star P \parallel Q) + (P \parallel \star Q)$

  $P \,|\, Q = Q \,|\, P$ and $P \,|\, (Q \,|\, R) = (P \,|\, Q) \,|\, R$.

- **Bounded** ! **and** $\star$: $!_I P \stackrel{\text{def}}{=} \prod_{i \in I} \mathbf{next}^i P$ and $\star_I P \stackrel{\text{def}}{=} \sum_{i \in I} \mathbf{next}^i P$

Power Saver Example

- A power saver :

$$!(\mathbf{unless}\,(\mathrm{lights} = \mathrm{off})\,\mathbf{next}\ \star\,\mathbf{tell}(\mathrm{lights} = \mathrm{off}))$$

- A refined power saver :

$$!(\mathbf{unless}\,(\mathrm{lights} = \mathrm{off})\,\mathbf{next}\ \star_{[0,60]}\,\mathbf{tell}(\mathrm{lights} = \mathrm{off}))$$

- A more refined one; deterministic power saver:

$$!(\mathbf{unless}\,(\mathrm{lights} = \mathrm{off})\,\mathbf{next}\,\mathbf{tell}(\mathrm{lights} = \mathrm{off}))$$

Operational Semantics

▷ **Internal Transitions:**

$$RT\ \frac{}{\langle \mathbf{tell}(c), a\rangle \longrightarrow \langle \mathbf{skip}, a \wedge c\rangle} \qquad RG\ \frac{a \vdash c_j}{\left\langle \sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i, a\right\rangle \longrightarrow \langle P_j, a\rangle}$$

$$RB\ \frac{}{\langle !\, P, a\rangle \longrightarrow \langle P \parallel \mathbf{next}\, !\, P, a\rangle} \qquad RS\ \frac{}{\langle \star\, P, a\rangle \longrightarrow \langle \mathbf{next}^n P, a\rangle}^{(n\geq 0)}$$

▷ **Observable Transition**

$$RO\ \frac{\langle P, a\rangle \longrightarrow^* \langle Q, a'\rangle \not\longrightarrow}{P \stackrel{(a,a')}{\Longrightarrow} \mathbf{F}(Q)} \quad = \begin{cases} Q' & \text{if } Q = \mathbf{next}\ Q' \\ Q' & \text{if } Q = \mathbf{unless}\ (c)\ \mathbf{next}\ Q' \\ \mathbf{F}(Q_1) \parallel \mathbf{F}(Q_2) & \text{if } Q = Q_1 \parallel Q_2 \\ \mathbf{local}\ x\ \mathbf{in}\ \mathbf{F}(Q') & \text{if } Q = \mathbf{local}\ x\ \mathbf{in}\ Q' \\ \mathbf{skip} & \text{otherwise} \end{cases}$$

Observations to Make of Processes

▷ **Stimulus-response interaction**

$$P = P_1 \xRightarrow{(c_1,c_1')} P_2 \xRightarrow{(c_2,c_2')} P_3 \xRightarrow{(c_3,c_3')} \dots$$

denoted by $P \xRightarrow{(\alpha,\alpha')}{}^{\omega}$ with $\alpha = c_1.c_2\dots$ and $\alpha' = c_1'.c_2'\dots$

Observable Behavior

▷ **Input-Output** $io(P) = \{(\alpha,\alpha') \mid P \xRightarrow{(\alpha,\alpha')}{}^{\omega}\}$

▷ **Output** $o(P) = \{\alpha' \mid P \xRightarrow{(\mathtt{true}^{\omega},\alpha')}{}^{\omega}\}$

▷ **Strongest Postcondition** $sp(P) = \{\alpha' \mid P \xRightarrow{(\_,\alpha')}{}^{\omega}\}$

Strongest-Postcondition Denotational Semantics

$$[\![\, \textbf{tell}(a) \,]\!] = \{c \cdot \alpha \in C^\omega \;:\; c \vdash a, \}$$

$$[\![\, P \parallel Q \,]\!] = [\![\, P \,]\!] \cap [\![\, Q \,]\!]$$

$$[\![\, !P \,]\!] = \{\alpha \;:\; \text{for all } \beta \in C^*, \alpha' \in C^\omega : \alpha = \beta.\alpha' \text{ implies } \alpha' \in [\![\, P \,]\!]\}$$

$$[\![\, \star P \,]\!] = \{\beta.\alpha \;:\; \beta \in C^*, \alpha \in [\![\, P \,]\!]\}$$

$$[\![\, \textstyle\sum_{i \in I} \textbf{when}\ (a_i)\ \textbf{do}\ P_i \,]\!] = \bigcup_{i \in I}\{c \cdot \alpha \;:\; c \vdash a_i \text{ and } c \cdot \alpha \in [\![\, P_i \,]\!])\cup \left(\bigcap_{i \in I}\{c \cdot \alpha \;:\; c \not\vdash a_i\,,\ \alpha \in C^\omega\}\right)$$

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, $sp(P) = [\![\, P \,]\!]$

A Logic à la Pnueli for ntcc

Syntax. $A := c \mid A \wedge A \mid \neg A \mid \exists_x A \mid \circ A \mid \diamond A \mid \Box A$

Semantics. Say $\alpha \models A$ iff $\langle \alpha, 1 \rangle \models A$ where

$\langle \alpha, i \rangle \models c$	iff	$\alpha(i) \vdash c$
$\langle \alpha, i \rangle \models \neg A$	iff	$\langle \alpha, i \rangle \not\models A$
$\langle \alpha, i \rangle \models A_1 \wedge A_2$	iff	$\langle \alpha, i \rangle \models A_1$ and $\langle \alpha, i \rangle \models A_2$
$\langle \alpha, i \rangle \models \circ A$	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 α' $x$variant of α s.t. $\langle \alpha', i \rangle \models A$.

Collection of all models: $[\![A]\!] = \{\alpha \mid \alpha \models A\}$

Satisfaction: $P \models A$ iff $sp(P) \subseteq [\![A]\!]$ (i.e., all outputs of P satisfy A)

Proof System for $P \models A$

$$\textbf{tell}(c) \vdash c \quad (\text{tell})$$

$$\frac{P \vdash A \quad Q \vdash B}{P \parallel Q \vdash A \wedge B}\ (\text{par}) \qquad \frac{P \vdash A}{\textbf{local}\ x\ \textbf{in}\ P \vdash \exists_x A}\ (\text{hide})$$

$$\frac{P \vdash A}{\textbf{next}\ P \vdash \circ A}\ (\text{next})$$

$$\frac{P \vdash A}{!P \vdash \Box A}\ (\text{rep}) \qquad \frac{P \vdash A}{\star P \vdash \Diamond A}\ (\text{star})$$

$$\frac{\forall i \in I \ \ P_i \vdash A_i}{\sum_{i \in I} \textbf{when}\ c_i\ \textbf{do}\ P_i \vdash \bigvee_{i \in I}(c_i \wedge A_i) \vee \bigwedge_{i \in I} \neg c_i}\ (\text{sum})$$

$$\frac{P \vdash A \quad A \Rightarrow B}{P \vdash B}\ (\text{rel})$$

Theorem. (**Completeness**) *For every P, A*

- ▷ $P \vdash A$ *implies* $P \models A$ *and*
- ▷ $P \models A$ *implies* $P \vdash A$, *if* P *is locally-independent.*

Applications: Cells

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

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

- The **exchange** operation $exch_f(x, y)$ models $\boxed{y := x \,;\, x := f(x)}$.

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

Example. $x : (3) \parallel y : (5) \parallel exch_7(x, y) \stackrel{\cdot}{\Longrightarrow} x : (7) \parallel y : (3)$.

Applications: Logic & Proof System at Work

Proposition. $$\boxed{exch_f(x,y) \ \vdash \ (x = v) \Rightarrow \bigcirc(x = f(v) \wedge y = v)}.$$

$$\frac{}{x : (g(w)) \vdash x = g(w)}\,Pr.1 \qquad \frac{}{y : (w) \vdash y = w}\,Pr.1$$

$$\frac{x : (g(w)) \vdash x = g(w) \qquad y : (w) \vdash y = w}{x : (g(w)) \parallel y : (w) \vdash x = g(w) \mathbin{\dot\wedge} y = w}\,LPAR$$

$$\frac{x : (g(w)) \parallel y : (w) \vdash x = g(w) \mathbin{\dot\wedge} y = w}{\mathbf{next}\ (x : (g(w)) \parallel y : (w)) \vdash \bigcirc(x = g(w) \mathbin{\dot\wedge} y = w)}\,LNEXT$$

$$\frac{\mathbf{next}\ (x : (g(w)) \parallel y : (w)) \vdash \bigcirc(x = g(w) \mathbin{\dot\wedge} y = w)}{\mathbf{tell}(change(x)) \parallel \mathbf{tell}(change(y)) \parallel \mathbf{next}(x : f(w) \parallel \ y : (w)) \vdash \bigcirc(x = g(w) \mathbin{\dot\wedge} y = w)}\,Lem.(3)$$

$$\frac{\forall w \in \mathcal{D}\ \ \mathbf{tell}(change(x)) \parallel \mathbf{tell}(change(y)) \parallel \mathbf{next}(x : f(w) \parallel \ y : (w)) \vdash \bigcirc(x = g(w) \mathbin{\dot\wedge} y = w)}{exch_f(x,y) \vdash \dot{\bigvee_{w\in\mathcal{D}}} (x = w \mathbin{\dot\wedge} \bigcirc(x = g(w) \mathbin{\dot\wedge} y = w)) \mathbin{\dot\vee} \dot{\bigwedge_{w\in\mathcal{D}}} \dot\neg x = w}\,LSUM$$

$$\frac{exch_f(x,y) \vdash \dot{\bigvee_{w\in\mathcal{D}}} (x = w \mathbin{\dot\wedge} \bigcirc(x = g(w) \mathbin{\dot\wedge} y = w)) \mathbin{\dot\vee} \dot{\bigwedge_{w\in\mathcal{D}}} \dot\neg x = w}{exch_f(x,y) \vdash \dot{\bigwedge_{w\in\mathcal{D}}} (x = w \mathbin{\dot\Rightarrow} \bigcirc(x = g(w) \mathbin{\dot\wedge} y = w))}\,LCONS$$

$$\frac{exch_f(x,y) \vdash \dot{\bigwedge_{w\in\mathcal{D}}} (x = w \mathbin{\dot\Rightarrow} \bigcirc(x = g(w) \mathbin{\dot\wedge} y = w))}{exch_f(x,y) \vdash (x = v \mathbin{\dot\Rightarrow} \bigcirc(x = g(v) \mathbin{\dot\wedge} y = v))}\,LCONS$$

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`.

$$
\begin{array}{llll}
\mathit{GoForward} & \stackrel{\text{def}}{=} & & \mathtt{f}_{exch}(act_1, act_2) \parallel \mathbf{tell}(\mathtt{forward}) \\
\mathit{GoRight} & \stackrel{\text{def}}{=} & & \mathtt{r}_{exch}(act_1, act_2) \parallel \mathbf{tell}(\mathtt{right}) \\
\mathit{GoLeft} & \stackrel{\text{def}}{=} & & \mathtt{l}_{exch}(act_1, act_2) \parallel \mathbf{tell}(\mathtt{left}) \\
\mathit{Zigzag} & \stackrel{\text{def}}{=} & (& \mathbf{when}\,(act_1 \neq \mathtt{f})\,\mathbf{do}\ \mathit{GoForward} \\
 & & + & \mathbf{when}\,(act_2 \neq \mathtt{r})\,\mathbf{do}\ \mathit{GoRight} \\
 & & + & \mathbf{when}\,(act_2 \neq \mathtt{l})\,\mathbf{do}\ \mathit{GoLeft}\ ) \\
 & & \parallel & \mathbf{next}\ \mathit{Zigzag} \\
\mathit{StartZigzag} & \stackrel{\text{def}}{=} & & act_1{:}(0) \parallel act_2{:}(0) \parallel \mathit{Zigzag}
\end{array}
$$

Proposition. $\mathit{StartZigzag} \vdash \Box(\Diamond\mathtt{right} \wedge \Diamond\mathtt{left})$

Behavioral Equivalences

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

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

Theorem. $\approx_{io} = \approx_o \subset \sim_{io} \subset \sim_o$.

Distinguishing Context Characterizations

Theorem. *Let* $\sim \in \{\approx_o, \sim_{io}, \sim_{sp}\}$*. One can construct contexts* $U[.]$ *and* $C_{\sim}^{(P,Q)}[.]$ *such that for all* P, Q*:*

▷ $P \approx_o Q$ *iff* $U[P] \sim_o U[Q]$ *(for finite set of constraints).*

▷ $P \sim Q$ *iff* $C_{\sim}^{(P,Q)}[P] \sim_o C_{\sim}^{(P,Q)}[Q]$*.*

- Interesting consequence of the theorem:

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

- Interesting result introduced for the proof of the theorem:

> 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.*

Variants and their Expressive Power

Deterministic ntcc with the following alternatives for ***infinite behavior.***

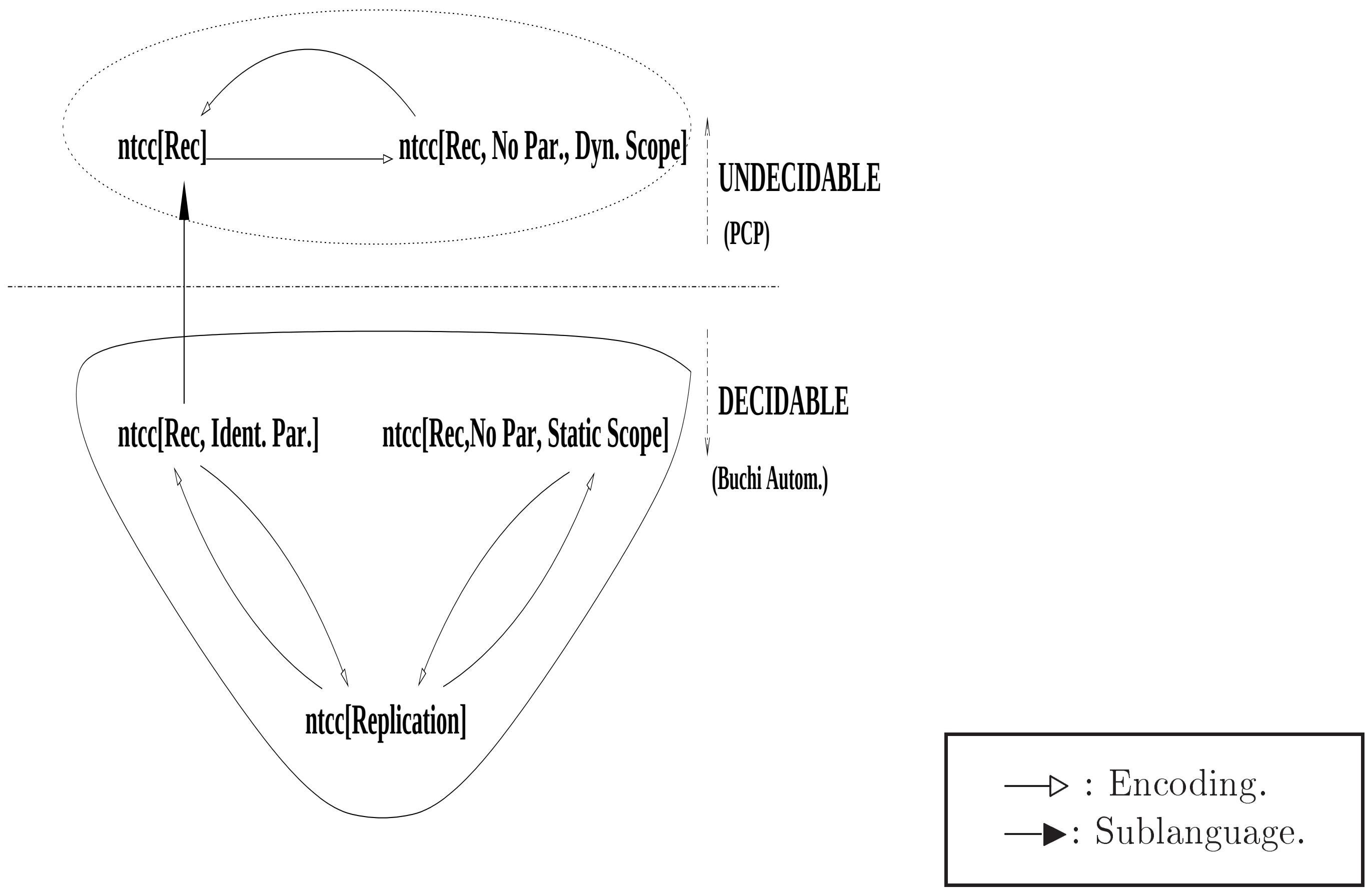
- **tcc[Rec]**
  Recursive definitions $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$ with $fv(P) \subseteq \{x_1, \ldots, x_n\}$.

- **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 $A \stackrel{\text{def}}{=} P$ with Dynamic Scoping

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

TCC Hierarchy and $\sim_{io}$ (un)decidability.



- The results clarify conjectures made in the literature.
- Qualitative distinction between dynamic and static scope.
- The results involve FSA, PCP, Encodings and Bisimulations.
- The results have inspired similar results for CCS.

Remarks and Future Work

We have presented

- ▷ ntcc; a calculus for discrete timed systems.
- ▷ Denotation, linear-time logic and proof system for ntcc.
- ▷ Examples illustrating the applicability of the calculus.
- ▷ Equivalences, congruence and (un)decidability results.
- ▷ Hierarchy of temporal CCP languages

Current and Future Work

- ▷ (Un)decidability results for the full calculus and process logic.
- ▷ Branching temporal logic for the calculus.
- ▷ Probabilistic extension of ntcc.
- ▷ Programming language for RCX controllers based on ntcc.
- ▷ ***The role of ntcc (and CCP) in modeling security protocols.***

Paper Contributions.

- **Book Chapter**.
  1. M. Nielsen and F. Valencia. *Temporal Concurrent Constraint Programming: Applications and Behavior*. Formal and Natural Computing: Essays Dedicated to Grzegorz Rozenberg. Springer, LNCS 2300. 2002.

- **Journal article**.
  2. M. Nielsen, C. Palamidessi and F. Valencia. *Temporal Concurrent Constraint Programming: Denotation, Logic and Applications*. Nordic Journal of Computing, Vol. 9. 2002.

- **Proceedings of International Conferences**.
  3. M, Nielsen, C. Palamidessi and F. Valencia. *On the Expressive Power of Concurrent Constraint Programming Languages*. In Proc. of PPDP 2002. ACM Press. 2002.

  4. C. Rueda and F. Valencia. *Proving musical properties using a temporal concurrent constraint calculus*. In Proc. of ICMC 2002. ICMC 2002.

5. F. Valencia. *Temporal Concurrent Constraint Programming* (Ext. Abstract). In Proc. of CP2001. Springer-Verlag, LNCS 2239. 2001.

6. C. Palamidessi and F. Valencia. *A Temporal Concurrent Constraint Programming Calculus*. In Proc. of CP2001. Springer-Verlag, LNCS 2239. 2001.

- **Workshops and Newsletters**.

7. Mogens Nielsen and Frank D. Valencia. *Temporal Concurrent Constraint Programming: A Framework for Discrete-Timed Systems*. Vol 15 n. 4 of the Association for Logic Programming (ALP) Newsletter. 2003

8. Camilo Rueda and Frank D. Valencia. *Formalizing Timed Musical Processes with a Temporal Concurrent Constraint Programming Calculus*. CP2001.

9. Mogens Nielsen, Catuscia Palamidessi and Frank D. Valencia. *A Calculus for Temporal Concurrent Constraint Programming*. EXPRESS'01. 2001.

Examples of Observables

$$\underbrace{\textbf{when } a \textbf{ do next } \begin{array}{l} \textbf{when } b \textbf{ do next tell}(d) \\ + \\ \textbf{when } c \textbf{ do next tell}(e) \end{array}}_{P} \quad , \quad \underbrace{\begin{array}{l} \textbf{when } a \textbf{ do next when } b \textbf{ do next tell}(d) \\ + \\ \textbf{when } a \textbf{ do next when } c \textbf{ do next tell}(e) \end{array}}_{Q}$$

Assuming a, b, c, d and e mutually exclusive:

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