music performance, micro-controllers). Agents are constrained by timed requirements. (E.g., people at a company, etc.)

Timed Systems:

Agents can change their communication links (E.g., mobile phones).

Mobile Systems:

Agents need to synchronize (E.g., phone calls).

Synchronous Systems:


Concurrent Systems:

Multiple agents (processes) that interact among each other (E.g., Internet).

Motivation: Concurrency and Time
The model we developed is the **nuc calculus**, the subject of this talk.

- defining domain specific programming languages.
- manipulating partial information
- specifying concurrency via constraints

**CCP** used for:

- specific domains applications
- partial information
- specifying concurrency
- concurrent behavior

**Timed Systems Involve:**

- A CCP model for describing and analyzing timed systems.

Motivation: Our Goal
Other Models: Guidelines
Synchrony.

Extends the computational model of tcc to allow for nondeterminism and

ntcc arises as a generalization of tcc (Saraswat et al., 94).

- Techniques from a denotational semantics and process logic.
- Formalization upon process algebra and logic.
- If express interesting real-world temporal situations.
- Simple ideas from concurrency and temporal logic.

ntcc is simple, expressive, formal and provides techniques.

Our Model: The nttc calculus.
How expressive are the constructs?

- $\mathbb{E}$: $\forall$ calculus, $\forall$ synchronous vs. asynchronous version.

- $\mathbb{E}$: $\forall$ logic for expressing process specifications (Hennessy-Milner logic)

How should process properties be specified and proved?

- (un)decidability.

How should processes be compared?

- $\mathbb{E}$: Observable behavior, process equivalences, congruences and their

- $\mathbb{E}$: Operational, denotational, or algebraic semantics

How should these constructs be endowed with meaning?

- $\mathbb{E}$: Atomic actions, parallelism, nondeterminism, hiding, recursion, etc.

Which process constructs fit the intended phenomenon?

Models for Concurrency: Key Issues Addressed.
How expressive are the ncc constructs?

•
E.g., expressive power hierarchy of variations of ncc.

E.g., Process logic and associated proof system.

•
How are ncc process properties specified and proved?

Behavioral equivalences, associated congruences and their (in)decidability:

•
How are ncc processes compared?

Operational and Denotational Semantics:

•
How are these constructs given meaning?

Non-determinism, replication, unbounded delays, unit delays and time-outs:

•
Which process constructs fit discrete-time systems?

Our Model: Key Issues Addressed.
One of the outstanding challenges in concurrency is to find the right marriage between logic and behavioral approaches.

Our Model: The main benefit.
That is, our work extends and strengthens the CCP theory of concurrency.

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 nict theory to study pre-existing CCP languages.

- (Un)Decidability results for their equivalences.
- First Temp. CCP expressive-power hierarchy

General Contributions
Agenda

The Rest of this Talk: Overview of our Work

February 2003
Synchronization via Blocking-Ask.

Concurrent Execution of Processes.

Partial Information (e.g. temperature is some unknown value < 20).

\[ \text{temperature} > 40 \]
\[ \text{temperature} = 30 \]
\[ \text{temperature} < 20 \]

CCP Initiations: A Typical CCP Scenario

Feb, 2003
Definition. A constraint system consists of a signature $\mathcal{S}$ and first-order theory $\Delta$ over $\mathcal{S}$.

$C \vdash \varphi$.

Set of constraints under consideration.

Decision: $\Delta$ decidable entailment relation between constraints.

$\mathcal{C}$.

Definition. Representing partial information.
micro-controllers in general.

Examples: Programmable Logic Controllers (PLC's), LEGO RCX bricks and

- **Stimulus-Response duration:** time interval (or time unit).
- **Response:** output information of $P_i$.
- **Stimulus:** input information for $P_i$.

\[
\begin{array}{cccccc}
\rightarrow & P_3 & \leftarrow & P_2 & \leftarrow & P_1 \\
\downarrow & c_1 & \leftarrow & c_2 & \leftarrow & c_1 \\
\downarrow & c_3 & \leftarrow & c_3 & \leftarrow & c_3 \\
\end{array}
\]
<table>
<thead>
<tr>
<th>Action within the time interval</th>
<th>Description</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>∅</strong> and <strong>p</strong> unless c now in the store or next <strong>p</strong></td>
<td>parallelism</td>
<td>**∅</td>
</tr>
<tr>
<td><strong>p</strong> delay one time unit</td>
<td>time-out</td>
<td><strong>p</strong> unless next <strong>p</strong></td>
</tr>
<tr>
<td>execute <strong>p</strong> with local <strong>x</strong></td>
<td>unit-delay</td>
<td><strong>p</strong> next <strong>p</strong></td>
</tr>
<tr>
<td>when c in the store execute <strong>p</strong></td>
<td>hiding</td>
<td><strong>p</strong> local x in <strong>p</strong></td>
</tr>
<tr>
<td>add c to the store</td>
<td>asking information</td>
<td><strong>p</strong> when c do <strong>p</strong></td>
</tr>
<tr>
<td>telling information</td>
<td>telling information</td>
<td>tell(∅)</td>
</tr>
</tbody>
</table>

**Ntc Syntax : Basic tcc Processes**

Feb, 2003
... \( \parallel p \parallel \text{next} \parallel p \parallel \text{next} \parallel p \\ldots \)

Unbounded many copies of \( p \) one at a time: \( p \)

### Infinite Behavior: \( p \)

Unbounded but finite delay of \( p \)

### Asynchronous Behavior: \( p \)

Guarded Choice:

### Non Deterministic Behavior: \( \nexists \text{when } \text{do } p \)

### Ntc Additional Basic Processes
Some Derived Constructs

\[ d_{\text{next}}^{I \in \mathcal{E}} \subseteq d^{I*} \quad \text{and} \quad d_{\text{next}}^{I \in \mathcal{E}} \bigcap_{d \in \mathcal{E}} d^{I_i} : * \quad \text{bounded} \]

\[ (R \mid (\mathcal{O} \mid d)) = (R \mid \mathcal{O}) \mid d \quad \text{and} \quad d \mid \mathcal{O} = \mathcal{O} \mid d \]

Fair asynchronous parallel:

\[ (\mathcal{O} \parallel d) + (\mathcal{O} \parallel d^*) = d \mid \mathcal{O} \]

Abortion: abort = abort \parallel d

Abortion: abort \in \{tell(false) \}

\[ d = \text{skip} \parallel d \]

Inactivity:

\[ d^{I \in \mathcal{E}} \subseteq \text{skip} \quad \text{for each} \quad d \]

Feb, 2003
A more refined one; deterministic power saver:

\[
\text{unless } \text{lights} = \text{off} \text{ [tell] next} \text{ lights} = \text{off} \text{ [off]}
\]

A refined power saver:

\[
\text{unless } \text{lights} = \text{off} \text{ [tell] next } \text{ lights} = \text{off} \text{ [off]}
\]

A power saver:

Power Saver Example
\[
\begin{align*}
\text{otherwise} & \\
\hat{\sigma} \in \text{local } x \text{ in } & \text{skip} \\
\hat{\sigma} \parallel \hat{\sigma}_1 \hat{\sigma} & = \hat{\sigma}_1 \\
\text{unless next } (\sigma) & = \hat{\sigma}_1 \\
\hat{\sigma}_1 \text{ next } & = \hat{\sigma} \\
\hat{\sigma}_1 & \\
& \\
= (\hat{\sigma}) P & \leftarrow (\tau, a) \quad p \\
\leftarrow \langle \tau, a \rangle & \leftarrow \langle \tau, a \rangle \\
\hat{\sigma} \parallel \text{next } (\sigma) & \text{ Observable Transition} \\
\text{RS} & \\
\langle a \rangle P & \leftarrow \langle a \rangle P \ast \\
\hat{\sigma} \parallel \text{next } & \text{ RB} \\
\langle a \rangle P & \\
\text{when } c \text{ do } P & \text{ RC} \\
\langle a \rangle & \leftarrow \langle a \rangle \\
\text{skip, a} & \leftarrow \langle a \rangle \\
\text{tell, c} & \\
\text{Internal Transitions:} \\
\text{Operational Semantics}
\end{align*}
\]
\[
\begin{align*}
\{ \nu \leftarrow_{(\nu, \lambda)} d \mid \nu \} = (d)_{ds} & \quad \text{Strongest Postcondition} \\
\{ \nu \leftarrow_{(\nu, \lambda)} d \mid \nu \} = (d)^o & \quad \text{Output} \\
\{ \nu \leftarrow_{(\nu, \lambda)} d \mid (\nu, \nu) \} = (d)^i & \quad \text{Input-Output} \\
\end{align*}
\]

Observable Behavior

\[ \cdots \leftarrow_{(c_3, c_3)} p_3 \leftarrow_{(c_2, c_2)} p_2 \leftarrow_{(c_1, c_1)} p_1 = d \]

\[ \text{Stimulus-Response Interaction} \]

Observations to Make of Processes
Theorem. $P$ is locally-independent if its guards depend on no local variables.

\[
\begin{align*}
\llbracket d \rrbracket &= (d) ds \\
\text{and, if } d \text{ is a locally-independent, } \llbracket d \rrbracket &\subseteq (d) ds
\end{align*}
\]

Definition. $P$ is locally-independent if its guards depend on no local variables.

\[
\begin{align*}
\llbracket d \rrbracket &= (d) ds \\
\text{and, if } d \text{ is a locally-independent, } \llbracket d \rrbracket &\subseteq (d) ds
\end{align*}
\]

Strongest-Postcondition Denotational Semantics

Feb. 2003

Temporal CCP
\[(\forall v \in \mathcal{D}) \iff \exists (p \in \mathcal{D}) | v \models p \quad \text{Satisfaction:}\]

\[
\{ v \models \phi | \phi \} = [v] \quad \text{Collection of all models:}\]

\[
\begin{align*}
\text{Semantics. Say } v & \models \phi \iff v \models \langle \emptyset, \phi \rangle \\
\text{Syntax. } A & : = \emptyset | \emptyset \sqcup A | \emptyset \cap A | A \land \neg A | A \lor A \land \neg A | A \land \neg A \land \neg A \land \neg A
\end{align*}
\]

A Logic à la Pnueli for ntcc
Proof System for $P \models A$

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

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

$\triangleright$ $P \vdash A$ implies $P \models A$ and

$\triangleright$ $P \models A$ implies $P \vdash A$, if $P$ is locally-independent.
Example.

\[
(\exists) : hi \quad \Leftrightarrow \quad (hi',x)_{exchange} \quad \exists (q) : hi \quad (\exists) : x
\]

\[
(\exists) : hi \quad \Leftrightarrow \quad (a)_{next} (hi)_{change} (x)_{tell} (\exists) \quad \exists (a = x)_{when} \quad \exists (a, x)_{exchange} \quad \exists (x)_{f}
\]

The exchange operation, exchange models (hi', x)_{change} for (a, x)_{exchange}.

\[
(x)_{f} = : x' \quad : x = : hi
\]

\[
(\exists) : x_{next} (x)_{change} \quad \exists (z = x)_{tell} \quad \exists (z) : x
\]

Cells: (a, x)_{denotes a cell x with contents u}.

Applications: Cells
\[
\cext \text{ CONS} \quad \frac{(a = \st{h} \vee (a) \b = x) \circ \iff a = x}{\text{ CONS}} \quad \frac{(m = \st{h} \vee (m) \b = x) \circ \iff m = x}{\text{ CONS}} \quad \frac{\forall \forall \st{h} \in \d \in m}{\text{ CONS}}
\]

\[
\text{Lem. (3)} \quad \frac{(m = \st{h} \vee (m) \b = x) \circ \iff ((m) : \st{h} \parallel (m) \f : x) \text{ next} \iff ((\text{change}) \text{tell}) \parallel ((\text{change}) \text{tell}) \parallel ((\text{change}) \text{tell})}{\forall \forall \st{h} \in \d \in m}
\]

\[
\text{Lem. (4)} \quad \frac{(m = \st{h} \vee (m) \b = x) \circ \iff ((m) : \st{h} \parallel ((m) \b) : x) \text{ next}}{\forall \forall \st{h} \in \d \in m}
\]

\[
\text{Prop.} \quad \frac{(a = \st{h} \vee (a) \f = x) \circ \iff (a = x) \iff (\st{h}, x) \f}{\forall \forall \st{h} \in \d \in m}
\]

Applications: Logic & Proof System at Work
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.

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

Proposition. \( \text{StartZigzag} \vdash \square(\Diamond \text{right} \land \Diamond \text{left}) \)
Theorem. \( \sim_0 \circ \sim_0 \subseteq \sim_0 \circ \sim_0 \).

Definition. Let \( l \in L \setminus \{0, 10, 50\} \). Define \( P \sim^1 P' \) iff \( P(l) = 1(0) \).

But neither \( \sim_0 \circ \sim_0 \) nor \( \sim_0 \circ \sim_0 \) are congruences. Let \( \sim_0 \) and \( \sim_0 \) be the corresponding

Behavioral Equivalences
Given one can construct a finite set including all relevant inputs.

Interesting result introduced for the proof of the theorem:

Decidability of all \( \equiv_0 \) reduce to that of

Interesting consequence of the theorem:

Theorem. Let \( \exists \in \sim_{d^s} \sim_0 \equiv_0 \sim_q \{1 \} \). One can construct contexts \( \Omega \) and

Distiguishing Context Characterizations

Feb. 2003

Temporal CCP
Theorem. As a corollary, given a locally-deterministic process $p$, one can effectively construct a B"uchi automaton $B_p$ that recognizes $o(p)$. Each of its states represent the state of $p$ at some point in time. However, the states associated with $p$'s self-loops and the states visited during the B"uchi acceptance condition do not occur outside of $p$'s local processes.

Definition. A star-free process $p$ is locally-deterministic if and only if all its summands are all decidable for locally-deterministic processes.

Behavioral Equivalence: Decidability.
Recursive definitions $A \defeq d \triangleright$ with Static Scoping

$\triangleright$ Rec, No Parameters, Static Scoping

Recursive definitions $A \defeq d \triangleright$ with Dynamic Scoping

$\triangleright$ Rec, No Parameters, Dyn. Scoping

As above but every call of $A$ in $D$ is of the form $A(x_1, \ldots, x_n)$.

$\triangleright$ Rec, Identical Parameters

$\{u x, \ldots, z x\} \subseteq (d) \land f$ with $d \defeq (u x, \ldots, z x)\triangleright A(x_1, \ldots, x_n)$

Recursive definitions $A \defeq d \triangleright$ with $D$.

$\triangleright$ Rec

Infinite behavior.

Deterministic nctc with the following alternatives for Variants and their Expressive Power.
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.
The role of ntc (and CCP) in modeling security protocols.

- Programming language for ntc controllers based on ntc.
- Probabilistic extension of ntc.
- Branching temporal logic for the calculus.

(UN)decidability results for the full calculus and process logic.

Current and Future Work

- Hierarchy of temporal CCP languages
- Equivalence, congruence and (UN)decidability results.
- Examples illustrating the applicability of the calculus.
- Denotation, linear-time logic and proof system for ntc.
- We have presented ntc, a calculus for discrete timed systems.

Remarks and Future Work
Paper Contributions.

- **Book Chapter.**

- **Journal article.**

- **Proceedings of International Conferences.**


- **Workshops and Newsletters.**


(D)ds \not\ni a \text{ but } a \in \{\text{true}\} \Rightarrow (O)ds \neq (D)ds \quad \bullet

(D)io \not\ni (a, a) \text{ but } (O)io \ni (a, a) \in \{\text{true}\} \Rightarrow (O)io \neq (D)io \quad \bullet

\{\text{true}\} = (O)o = (D)o \quad \bullet

Assuming a, q, c, p and e mutually exclusive:

\begin{align*}
\text{when } a \text{ do } \text{next tell } e \\
\text{when } c \text{ do } \text{next tell } e \\
\text{when } q \text{ do } \text{next tell } e
\end{align*}

Examples of Observables