Model Checking for tcc Calculus

Jaime E. Arias Almeida

Supervisors:
Dr. Carlos Olarte
Dr. Eugenio Tamura

AVISPA Research Group
Pontificia Universidad Javeriana - Cali

December, 2012
Motivation

Concurrent systems are everywhere:

- **Engineering:** Security protocols, mobile computing, synchronous systems.
- **Science:** Biological and chemical systems.
- **Arts:** Multimedia interaction.

Models of Concurrency

Formal models to describe and analyze concurrent systems. They must be:

- Simple.
- Expressive.
- Formal.
- Provide reasoning techniques.

Some examples: CCS [Mil89], the $\pi$ calculus [Mil99, SW03], CSP [Hoa85], CCP [Sar93].
Motivation

- Concurrent Constraint Programming (ccp) [Sar93] is a formalism for concurrency where agents interact with one another by adding (telling) and reading (asking) information represented as constraints in a shared medium (global store).

- The type of constraints and the entailment relation is given by a Constraint System (e.g. $x > 42 \models x > 0$).
Motivation

- Temporal Concurrent Constraint Programming (tcc) [SJG94] extends ccp by adding temporal constructs for modeling timed and reactive systems.
- Formal verification plays an important role in detecting errors in concurrent systems. The presence of a system failure in air traffic control systems, medical instruments, aircrafts or in general, reactives systems could be catastrophic.

Figure: Some disasters

(a) Therac-25  

(b) Ariane-5
Motivation

- Verification of tcc programs is performed using inductive techniques since the model does not provide automatic formal verification tools.
- Problems:
  - Difficult
  - Error prone
  - Experts are required.
- Hubert Garavel [Gar08]: “...The research agenda for theoretical concurrency should therefore address the design of efficient algorithms for translating and verifying formal specifications of concurrent systems”. 

Motivation

• Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model [CGP99].

• Advantages
  ○ Formal technique.
  ○ Fully-automatic.
  ○ Experts are not required.
  ○ Counterexample.

• Drawback:
  ○ State-space explosion problem.
Our Goal

• We aim at developing a model checking algorithm for tcc calculus to automatically verify digital systems.
Our approach: A model checking algorithm based on the work of Falaschi and Villanueva [FV06].
Our Contributions

- A structure for modeling the behavior of a tcc program.
- An algorithm for constructing the model from the tcc specification.
- A method for reducing the size of a model.
- A model checking algorithm for verifying tcc programs.
- A prototype of the model checking algorithm.
Outline

1. The tcc Calculus
2. The Model Checking Technique
3. A Model for tcc Programs
4. A Logic for the Specification of Properties
5. The Model Checking Algorithm
6. A Prototypical Tool
7. Application
8. Concluding Remarks
Outline

1. The tcc Calculus
2. The Model Checking Technique
3. A Model for tcc Programs
4. A Logic for the Specification of Properties
5. The Model Checking Algorithm
6. A Prototypical Tool
7. Application
8. Concluding Remarks
The tcc Calculus [SJG94]
Intuitive Description and Operational Semantics

温度 = ?

ask (temperature < 100) then Q

tell (temperature > 45)

温度 = ?
tell (temperature < 70)

ask (temperature = 50) then P
The tcc Calculus [SJG94]
Intuitive Description and Operational Semantics

\[
\text{ask (temperature < 100) then Q}
\]

\[
\text{tell (temperature > 45)} \quad \text{temperature} = ? \quad \text{tell (temperature < 70)}
\]

\[
\text{ask (temperature = 50) then P}
\]
ask (temperature < 100) then Q

42 < temperature < 70

ask (temperature = 50) then P
The \texttt{tcc} Calculus [SJG94]
Intuitive Description and Operational Semantics

\begin{align*}
\text{ask} & \left( \text{temperature} < 100 \right) \text{ then } Q \\
42 < \text{temperature} < 70 \\
\text{ask} & \left( \text{temperature} = 50 \right) \text{ then } P
\end{align*}
The tcc Calculus [SJG94]
Intuitive Description and Operational Semantics

\begin{align*}
&\text{ask } (\text{temperature} = 50) \text{ then } P \\
&42 < \text{temperature} < 70 \\
&\text{Q} \\
&\text{Remains Blocked}
\end{align*}
The tcc Calculus [SJG94]
Intuitive Description and Operational Semantics

1. Receives a stimulus (i.e. a constraint) from the environment.
1. Receives a stimulus (i.e. a constraint) from the environment.
2. Computes a ccp process in the current time unit and wait for stability.
1. Receives a stimulus (i.e. a constraint) from the environment.
2. Computes a ccp process in the current time unit and wait for stability.
3. Responds with the resulting store.
1. Receives a stimulus (i.e. a constraint) from the environment.
2. Computes a ccp process in the current time unit and wait for stability.
3. Responds with the resulting store.
4. Executes the residual process in the next time unit.

* Note: Stores are not automatically transferred from a time unit to the next one.
## The tcc Calculus [SJG94]

### Syntax

<table>
<thead>
<tr>
<th>Process</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 in the current time interval</td>
</tr>
<tr>
<td><code>now c then P</code></td>
<td>asking information</td>
<td>executes $P$ if $c$ can be deduced from the current store</td>
</tr>
<tr>
<td>$\exists x(P)$</td>
<td>hiding</td>
<td>executes $P$ with local $x$</td>
</tr>
<tr>
<td>$P \parallel Q$</td>
<td>parallelism</td>
<td>executes $P$ and $Q$</td>
</tr>
<tr>
<td><code>next P</code></td>
<td>unit-delay</td>
<td>executes $P$ in the next time unit</td>
</tr>
<tr>
<td><code>now c else P</code></td>
<td>time-out</td>
<td>executes $P$ in the next time unit if $c$ cannot be entailed now</td>
</tr>
</tbody>
</table>
The tcc Calculus [SJG94]

Example

\[
p() ::=
\begin{align*}
  \text{now (in=\text{true}) then next(tell(x=2))} & \mid \\
  \text{now (in=\text{true}) else tell(x=1) } & \mid \ 	ext{next(p())}
\end{align*}
\]
Outline

1. The tcc Calculus
2. The Model Checking Technique
3. A Model for tcc Programs
4. A Logic for the Specification of Properties
5. The Model Checking Algorithm
6. A Prototypical Tool
7. Application
8. Concluding Remarks
**System Model:** Models of systems describe the behavior of systems in a formal way. They are mostly expressed using transition systems.
Property Specification: Properties should be described in a precise and unambiguous manner. Temporal logic is typically used to express properties.
The Model Checking Technique

Model Checking: Algorithm which checks the validity of the property in all states of the system model.
Outline

1. The tcc Calculus
2. The Model Checking Technique
3. A Model for tcc Programs
4. A Logic for the Specification of Properties
5. The Model Checking Algorithm
6. A Prototypical Tool
7. Application
8. Concluding Remarks
A Model for tcc Programs

Program Labeling

- The labeling process consists in assigning a different label to each occurrence of an agent in the program.
- Labels provide an unique identification for agents in order to know the point of execution of the program during the construction of the model.

Example

\[
\{l_{p_0}\} p() ::
\{l_{||_1}\}(\{l_{nwp_2}\} now (in=true) \text{ then } \{l_{next_3}\} next(\{l_{tell_4}\} tell(x=2)) || \\
\{l_{||_5}\}(\{l_{nown_6}\} now (in=true) \text{ else } \{l_{tell_7}\} tell(x=1) || \\
\{l_{next_8}\} next(\{l_{p_9}\} p()))
\]
A Model for tcc Programs

The tcc Structure

• **tcc Structure** is a graph structure which allows to model the behavior of a tcc specification.

• It is a variant of a **Kripke Structure**.

![Diagram](attachment:figure.png)

**Figure**: Components of a tcc Structure
A Model for tcc Programs

The tcc Structure

Definition (tcc Structure)

Let $AP$ be a set of atomic propositions, we define a tcc Structure $M$ over $AP$ as a seven tuple $M = (S, S_0, T, R, C, L, LT)$ where

1. $S$ is a finite set of states.
2. $S_0 \subseteq S$ is the set of initial states.
3. $T = \{i, t\}$ is the set of possible type of transitions. $i$ denotes an internal transition while $t$ denotes a temporal transition.
4. $R \subseteq S \times S \times T$ is a transition relation.
5. $C : S \rightarrow 2^{AP}$ is the function that returns the set of atomic propositions in a given state.
6. $L : S \rightarrow 2^L$ is the function that returns the set of internal labels in a given state.
7. $LT : S \rightarrow 2^L$ is the function that returns the set of temporal labels in a given state.
A Model for tcc Programs

Construction of the Model

- Transitions are described according to the operational rules of each tcc process.
- For each different declaration we construct a tcc Structure.
- Each state is composed of labels associated with agents that can be executed in a step of the construction process.
- Each label can be active or disable.
- The labels associated with temporal agents cannot be executed before all the labels associated with normal agents are executed.
A Model for tcc Programs

Construction of the Model

Procedure Call:

\[ p_0 \]

\[ \{ l_0 \} p() :: \\
\{ l_1 \} \{ l_{\text{now}p_2} \text{now } (in=true) \text{ then } l_{\text{next}3} \text{next } l_{\text{tell}4} \text{tell } (x=2)) \mid \\
\{ l_5 \} \{ l_{\text{nown}6} \text{now } (in=true) \text{ else } l_{\text{tell}7} \text{tell } (x=1) \mid l_{\text{next}8} \text{next } (l_{p_0} p()) \} \]
A Model for tcc Programs

Construction of the Model

Procedure Call:

\[
\{l_{p0}\}p()::
\begin{align*}
&\{l_{\|1}\}\left(\{l_{\text{hownp2}}\}\text{now (in=true) then } \{l_{\text{next3}}\}\text{next} (\{l_{\text{tell4}}\}\text{tell}(x=2)) \right) \right| \\
&\{l_{\|5}\}\left(\{l_{\text{hown6}}\}\text{now (in=true) else } \{l_{\text{tell7}}\}\text{tell}(x=1) \right) \right| \{l_{\text{next8}}\}\text{next} (\{l_{p9}\}p())
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Procedure Call:
A Model for tcc Programs

Construction of the Model

Procedure Call:

{p0}p()::
   {||1}{nowp2}now (in=true) then {next3}{tell4}tell(x=2) ||
   {||5}{nown6}now (in=true) else {tell7}tell(x=1) || {next8}next({p9}p())
A Model for tcc Programs

Construction of the Model

Parallel:

\[
\begin{align*}
\{p_0\} p() :: \\
\{1\} (\{\text{now} p_2\} \text{now (in=true) then } \{\text{next}_3\} \text{next (tell(x=2)) } || \\
\{5\} (\{\text{nown}_{6}\} \text{now (in=true) else } \{\text{tell}_7\} \text{tell(x=1) } || \{\text{next}_8\} \text{next (p()')} ))
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Parallel:

\[
\{p_0\} p() ::
\{||1\}(\{\text{now}_{p2}\} \text{now (in=true) then }\{\text{next}_{3}\} \text{next(}\{\text{tell}_{4}\} \text{tell(x=2))} ||
\{||5\}(\{\text{nown}_{6}\} \text{now (in=true) else }\{\text{tell}_{7}\} \text{tell(x=1) }|| \{\text{next}_{8}\} \text{next(}\{p_9\} p())\)\]
A Model for \(\text{tcc}\) Programs

Construction of the Model

Parallel:

\[
\begin{align*}
\{p_0\} p() &::= \\
\{1\}_1 &\cdot (\{\text{nown}_{2}\} \text{now} \ (\text{in} = \text{true}) \text{ then } \{\text{next}_3\} \text{next}(\{\text{tell}_4\} \text{tell}(x=2)) \mid \\
\{5\}_5 &\cdot (\{\text{nown}_6\} \text{now} \ (\text{in} = \text{true}) \text{ else } \{\text{tell}_7\} \text{tell}(x=1) \mid \{\text{next}_8\} \text{next}(\{p_0\} p()))
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Parallel:

```
{\l p0} p() ::
  {l || 1} (\{nowp2\} now (in=true) then \{next3\} next(\tell 4\})tell(x=2) ||
  {l || 5} (\{nown6\} now (in=true) else \{tell7\} tell(x=1) || \{next8\} next(\{p9\} p())))
```
A Model for tcc Programs

Construction of the Model

Timed Positive Ask:

\[
\{l_{p0}\}p() ::
\{l_1\}(\{nowp2\}now \ (in=true) \ then \ \{l_{next3}\}next(\{tell4\}tell(x=2)) \ ||
\{l_5\}(\{nown6\}now \ (in=true) \ else \ \{l_{tell7}\}tell(x=1) \ || \ \{l_{next8}\}next(\{l_{p9}\}p()))
\]
Timed Positive Ask:

\[
\{ l_{p0} \} p()::
\begin{align*}
&\{ l_1 \}\left( \{ l_{\text{nowp}2} \} \text{now (in=true)} \text{ then } \{ l_{\text{next}3} \} \text{next (\{ l_{\text{tell}4} \} \text{tell(x=2)})} \right) \text{ \mid} \text{ \mid} \\
&\{ l_5 \}\left( \{ l_{\text{nown}6} \} \text{now (in=true) else } \{ l_{\text{tell}7} \} \text{tell(x=1) \mid \{ l_{\text{next}8} \} \text{next (\{ l_{p9} \} p())} \right)
\end{align*}
\]
A Model for \texttt{tcc} Programs

Construction of the Model

Timed Positive Ask:

\[
\{l_p_0\}_p() ::
\{l_1||\}
(\{l_{\text{nowp}_2}\}_{\text{now}}(\text{in=true}) \text{ then } \{l_{\text{next}_3}\}_{\text{next}}(\{l_{\text{tell}_4}\}_{\text{tell}}(x=2)) || \\
\{l_5||\}
(\{l_{\text{nown}_6}\}_{\text{now}}(\text{in=true}) \text{ else } \{l_{\text{tell}_7}\}_{\text{tell}}(x=1) || \{l_{\text{next}_8}\}_{\text{next}}(\{l_p\}_p())
\]
A Model for tcc Programs

Construction of the Model

Timed Positive Ask:

\[
\{l_{p0}\}p() ::
\begin{align*}
\{l_1\}(\{\text{now}\ p_2\} \text{now } (\text{in=}\text{true}) \text{ then } \{l_{\text{next}3}\} \text{next}\{l_{\text{tell}4}\} \text{tell}(x=2) \} \mid \\
\{l_5\}(\{\text{now}\ p_6\} \text{now } (\text{in=}\text{true}) \text{ else } \{l_{\text{tell}7}\} \text{tell}(x=1) \mid \{l_{\text{next}8}\} \text{next}\{l_{p9}\} p())
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Timed Positive Ask:

\[
\{l_p\}_p() ::
\]

\[
\{l_1\} (\{\text{now} \_p2\} \text{now} \ (in=true) \text{ then } \{l_{\text{next}3}\} \text{next} (\{t_{\text{tell4}}\} \text{tell} (x=2)) ||
\]

\[
\{l_{\text{next}5}\} (\{\text{nown6}\} \text{now} \ (in=true) \text{ else } \{t_{\text{tell7}}\} \text{tell} (x=1) || \{l_{\text{next}8}\} \text{next} (\{l_p\} \text{p}()))
\]
A Model for tcc Programs

Construction of the Model

Timed Positive Ask:

\[
\{ \text{l}_p^0 \} \text{p}() ::
\begin{align*}
& \text{||} _1 (\text{now}_{\text{false}} \text{ then } \text{next}_3 \text{tell}_{\text{false}}(x=2) \mid \mid \\
& \text{||} _5 (\text{now}_{\text{false}} \text{ else } \text{tell}(x=1) \mid \mid \text{next}_8 \text{p}())
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Timed Positive Ask:

\[
\{l_p_0\} p() ::
\{l_1\} (\{l_{\text{now}p_2}\} \text{now (in=true)} \text{ then } \{l_{\text{next}3}\} \text{next (}\{l_{\text{tell}4}\} \text{tell (x=2)})) \mid \mid
\{l_5\} (\{l_{\text{nown}n_6}\} \text{now (in=true)} \text{ else } \{l_{\text{tell}7}\} \text{tell (x=1)} \mid \mid \{l_{\text{next}8}\} \text{next (}\{l_{p_9}\} p())
\]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\{ p_0 \} p()::
\{ || \}_{1} \{ || \}_{1} \{ || \}_{1} \{ || \}_{1} \{ p_{\text{nowp2}} \} \text{now} (\text{in=true}) \text{ then } \{ || \}_{1} \{ \text{next3} \} \text{next} (\{ || \}_{1} \{ \text{tell4} \} \text{tell}(x=2)) \text{ || }
\{ || \}_{1} \{ || \}_{1} \{ \text{nown6} \} \text{now} (\text{in=true}) \text{ else } \{ \text{tell7} \} \text{tell}(x=1) \text{ || } \{ || \}_{1} \{ \text{next8} \} \text{next} (\{ || \}_{1} \{ p_9 \} p())
\]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\{p_0\}p():=
\begin{align*}
\{l_1\}(&\{\_\text{now,}\_\text{nown6,}\_\text{next8}\}_\text{now}(\text{in=true}) \text{ then } \{l_3\}_\text{next}(\{l_4\}_\text{tell}(x=2)) | | \\
\{l_5\}(&\{\_\text{nown6}\}_\text{now}(\text{in=true}) \text{ else } \{l_7\}_\text{tell}(x=1) | | \{l_8\}_\text{next}(\{l_9\}_p())) 
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\text{Timed Negative Ask:}
\]

\[
\{p_0\} p() ::
\{l_1\} (\{h_{\text{nwp2}}\} \text{now (in=true) then } \{l_3\} \text{next}\{l_{\text{tell4}}\} \text{tell(x=2)} ||
\{l_5\} (\{h_{\text{nwn6}}\} \text{now (in=true) else } \{l_7\} \text{tell(x=1)} || \{l_{\text{next8}}\} \text{next}\{l_{p9}\} p())
\]
A Model for \texttt{tcc} Programs

Construction of the Model

Timed Negative Ask:

\[
\{p_0\} p()::
\begin{array}{l}
\{l \mid 1\}(\{l_{\text{nowp2}}\} \text{now } (in=true) \text{ then } \{l_{\text{next3}}\} \text{next } (\{l_{\text{tell4}}\} \text{tell}(x=2)) || \\
\{l \mid 5\}(\{l_{\text{nown6}}\} \text{now } (in=true) \text{ else } \{l_{\text{tell7}}\} \text{tell}(x=1) || \{l_{\text{next8}}\} \text{next } (\{l_{p9}\} p()))
\end{array}
\]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\{p_0\} p():: \\
\{l_1\}\{l_{\text{nowp}2}\}\text{now (in=true) then } \{l_{\text{next}3}\}\text{next(}\{l_{\text{tell}4}\}\text{tell(x=2)) ||} \\
\{l_5\}\{l_{\text{nown}6}\}\text{now (in=true) else } \{l_{\text{tell}7}\}\text{tell(x=1) || } \{l_{\text{next}8}\}\text{next(}\{l_{p9}\}\text{p())}\]

A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[ \{ l_{p0} \} p() :: \]
\[ \{ l_{||1} \}(\{ l_{\text{nowp2}} \text{ now (in=true) then} \{ l_{\text{next3}} \text{ next (\{ l_{\text{tell4}} \text{ tell (x=2)} \})} || \\
\{ l_{||5} \}(\{ l_{\text{nown6}} \text{ now (in=true) else} \{ l_{\text{tell7}} \text{ tell (x=1)} || \{ l_{\text{next8}} \text{ next (\{ l_{p9} \} p())} \})) \]

\[ \{ l_{\text{in=true}} \} 4 \]
\[ \{ l_{\text{in=true}} \} 5 \]
\[ \{ l_{\text{in=true}} \} 19 \]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\{ l_0 \} p() ::
\{ l_1 \} (\{ l_{\text{nowp2}} \} \text{now (in=true)} \text{ then } \{ l_{\text{next3}} \} \text{next(\{ l_{\text{tell4}} \} \text{tell(x=2)})} \text{ } ||
\{ l_5 \} (\{ l_{\text{nown6}} \} \text{now (in=true)} \text{ else } \{ l_{\text{tell7}} \} \text{tell(x=1)} \text{ } || \{ l_{\text{next8}} \} \text{next(\{ l_{\text{p9}} \} p())})
\]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\{p_0\}p()::
\{l\l_1\}\{\{l_{\text{nowp2}}\}\text{now (in=true) then }\{l_{\text{next3}}\}\text{next (}\{l_{\text{tell4}}\}\text{tell (x=2))}\ | | \\
\{l\l_5\}\{\{l_{\text{nown6}}\}\text{now (in=true) else }\{l_{\text{tell7}}\}\text{tell (x=1) }| | \{l_{\text{next8}}\}\text{next (}\{l_{p9}\}\text{p()})\}
\]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\{p_0\} p()::
\begin{align*}
\{1\} (\{n_{nowp_2}\} \text{now (in=true) then } \{3\} \text{next (tell(x=2))} || \\
\{5\} (\{n_{nown_6}\} \text{now (in=true) else } \{7\} \text{tell(x=1)} || \{8\} \text{next (p())})
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\{ l_{p0} \} p() ::
\{ l \| l_1 \} (\{ l_{\text{nown}_2} \} \text{now} (\text{in=}true) \text{ then } \{ l_{\text{next}_3} \} \text{next} (\{ l_{\text{tell}_4} \} \text{tell}(x=2)) \| \\
\{ l \| l_5 \} (\{ l_{\text{nown}_6} \} \text{now} (\text{in=}true) \text{ else } \{ l_{\text{tell}_7} \} \text{tell}(x=1) \| \{ l_{\text{next}_8} \} \text{next} (\{ l_{p9} \} p()))
\]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\{ p_0 \} p() :: \{ \|_1 \} (\{ h_{nwp2} \} \text{now } (\text{in=true}) \text{ then } \{ h_{next3} \} \text{next}(\{ t_{tell4} \} \text{tell}(x=2)) \| \{ \|_5 \} (\{ h_{nown6} \} \text{now } (\text{in=true}) \text{ else } \{ t_{tell7} \} \text{tell}(x=1) \| \{ h_{next8} \} \text{next}(\{ p_9 \} p()))
\]
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

$$\{ p_0 \} p()::$$

$$\{ l\| l \} (\{ l_{\text{nownp2}} \} \text{now (in=true)} \text{ then } \{ l_{\text{next3}} \} \text{next (tell(x=2)) ||}$$

$$\{ l\| l \} (\{ l_{\text{nown6}} \} \text{now (in=true)} \text{ else } \{ l_{\text{tell7}} \} \text{tell(x=1)} || \{ l_{\text{next8}} \} \text{next (p())})$$
A Model for tcc Programs

Construction of the Model

Timed Negative Ask:

\[
\begin{align*}
&\{l_{p0}\} p() :: \\
&\{l||1\} (\{l_{nown6}\} now (in=true) then \{l_{next3}\} next (\{l_{tell4}\} tell(x=2)) || \\
&\{l||5\} (\{l_{nown6}\} now (in=true) else \{l_{tell7}\} tell(x=1) || \{l_{next8}\} next (\{l_{p9}\} p()))
\end{align*}
\]
A Model for \textit{tcc} Programs

Construction of the Model

\textbf{Timed Negative Ask:}

\[
\{!p_0 \}\ p() ::
\begin{align*}
&\{!||_{1} \}\{!n_{\text{nowp}2}\} \text{now (in=true) then } \{!||_{3} \}\text{next} (\{!t_{\text{tell}4}\}\text{tell} (x=2)) \ | \\
&\{!||_{5} \}\{!n_{\text{nown}6}\} \text{now (in=true) else } \{!||_{7} \}\text{tell} (x=1) \ | \ \{!||_{8} \}\text{next} (\{!p_9 \}\ p())
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Unit Delay:

\[
\{ p_0 \} p() ::=
\begin{align*}
&\{ l \| 1 \} (\{ l_{\text{nowp}_2} \} \text{now (in=true) then } \{ l_{\text{next}_3} \} \text{next (} \{ l_{\text{tell}_4} \} \text{tell(x=2)}) || \\
&\{ l \| 5 \} (\{ l_{\text{nown}_6} \} \text{now (in=true) else } \{ l_{\text{tell}_7} \} \text{tell(x=1) || } \{ l_{\text{next}_8} \} \text{next (} \{ l_{p_9} \} p()) )
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Unit Delay:

\[
\begin{align*}
\{p_0\} p() &::= \\
\{||_1\} (\{\text{now}\}_{p_2} \text{now (in=true) then } \{\text{next}_3\} \text{next (\{tell\}_{4} \text{tell(x=2)} \|)} \\
\{||_5\} (\{\text{nown}\}_{6} \text{now (in=true) else } \{\text{tell}\}_{7} \text{tell(x=1) } \| \{\text{next}_8\} \text{next (\{p_9\} p())})
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Unit Delay:

\[
\begin{align*}
\{p_0\} p ():: & \\
& \{l_1\} (\{l_{\text{nowp2}}\} \text{now (in=true) then } \{l_{\text{next3}}\} \text{next(\{l_{\text{tell4}}\} \text{tell(x=2)}) | |} \\
& \{l_5\} (\{l_{\text{nown6}}\} \text{now (in=true) else } \{l_{\text{tell7}}\} \text{tell(x=1) | | } \{l_{\text{next8}}\} \text{next(\{l_{p9}\} p()))})
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Unit Delay:

\[
\{ \text{p0} \} \text{p}() ::
\]
\[
\{ | | \}_1 (\{ \text{nowp2} \} \text{now} (\text{in=true}) \text{ then } \{ \text{next3} \} \text{next}(\{ \text{tell4} \} \text{tell}(x=2)) \} | | \]
\[
\{ | | \}_5 (\{ \text{nown6} \} \text{now} (\text{in=true}) \text{ else } (\{ \text{tell7} \} \text{tell}(x=1) | | \{ \text{next8} \} \text{next}(\{ \text{p9} \} \text{p}())))
\]
A Model for \texttt{tcc} Programs

Construction of the Model

Unit Delay:

\[
\{p_0\}p()::
\begin{align*}
\{\|1\}\{h_{\text{nowp}_2}\}\text{now (in=true) then } &\{\|\text{next}_3\}\text{next(}\{t_{\text{tell}_4}\}\text{tell(x=2)}) || \\
\{\|\text{even}\}\{\|h_{\text{nown}_6}\}\text{now (in=true) else } &\{\|\text{tell}_7\}\text{tell(x=1) || }\{\|\text{next}_8\}\text{next(}\{p_9\}p())
\end{align*}
\]}
A Model for tcc Programs

Construction of the Model

Passage to the Next Time:

\[
\begin{align*}
\{p_0\}p():: & \\
\{||1\}(\{\text{now}_{p_2}\}\text{now}(in=true) & \text{then } \{\text{next}\}_{p_3}\text{next}(\{\text{tell}\}_{p_4}\text{tell}(x=2)) \mid \\
\{||5\}(\{\text{nown}_{p_6}\}\text{now}(in=true) & \text{else } \{\text{tell}\}_{p_7}\text{tell}(x=1) \mid \{\text{next}\}_{p_8}\text{next}(\{p_9\}p()))
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Passage to the Next Time:

\[
\begin{align*}
\{p_0\} p() :& = \\
\{l_1\} (\{l_{\text{howp2}}\} \text{now (in=true) then } \{l_{\text{next3}}\} \text{next(\{l_{\text{tell4}}\} \text{tell(x=2)} \}) || \\
\{l_5\} (\{l_{\text{nown6}}\} \text{now (in=true) else } \{l_{\text{tell7}}\} \text{tell(x=1) || } \{l_{\text{next8}}\} \text{next(\{l_{p9}\} p())})
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Passage to the Next Time:

\[
\begin{align*}
\{p_0\} \mathbf{p}() &::= \\
\{l_1\} &\mid \{l_2\} &\mid \{l_5\} \\
&\{l_1\} (\{h_{\text{nowp2}}\} \text{now (in=true) then } \{l_{\text{next3}}\} \text{next (}\{t_{\text{tell4}}\} \text{tell (x=2)}) || \\
&\{l_2\} (\{h_{\text{nown6}}\} \text{now (in=true) else } \{t_{\text{tell7}}\} \text{tell (x=1) || } \{l_{\text{next8}}\} \text{next (}\{p_9\} \mathbf{p}())
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Passage to the Next Time:

\[
\begin{align*}
\{l_{p0}\}p():: \\
\{l_1\}(\{l_{howp2}\} & \text{now (in=true) then } \{l_{next3}\} \text{next(}\{l_{tell4}\}\text{tell(x=2)} \} || \\
\{l_5\}(\{l_{nown6}\} & \text{now (in=true) else } \{l_{tell7}\}\text{tell(x=1)} \} || \{l_{next8}\} \text{next(}\{l_{p9}\}p())
\end{align*}
\]
A Model for tcc Programs

Construction of the Model

Equivalent States:

• Reduction of states is necessary to decrease the state explosion problem.
• The store on the quiescent point is defined as the output of the computation.
A Model for tcc Programs

Construction of the Model

Equivalent States:

- Reduction of states is necessary to decrease the state explosion problem.
- The store on the quiescent point is defined as the output of the computation.
A Model for tcc Programs

Construction of the Model

Equivalent States:

- Reduciton of states is necessary to decrease the state explosion problem.
- The store on the quiescent point is defined as the output of the computation.
A Model for tcc Programs

Construction of the Model

Equivalent States:

- Reduction of states is necessary to decrease the state explosion problem.
- The store on the quiescent point is defined as the output of the computation.
A Model for tcc Programs

Simplification of the Model

(a) Reduction of a sequence of internal transitions

(b) Reduction of a branching

Figure: Reduction rules
Outline

1. The tcc Calculus
2. The Model Checking Technique
3. A Model for tcc Programs
4. A Logic for the Specification of Properties
5. The Model Checking Algorithm
6. A Prototypical Tool
7. Application
8. Concluding Remarks
A Logic for the Specification of Properties

The ntcc Logic

- Linear temporal logic (LTL) named CLTL for reasoning about ntcc processes.
- It expresses properties over sequences of constraints.
- ntcc is an extension of tcc.

Definition (CLTL Syntax)

The formulae $F, G, \ldots \in \mathcal{F}$ are built from constraints $c \in C$ in the underlying constraint system by

$$F, G, \ldots := c | \text{true} | \text{false} | F \land G | F \lor G | \neg F | \exists x F | \circ F | \square F | \diamond F$$

Semantics
Outline

1. The tcc Calculus
2. The Model Checking Technique
3. A Model for tcc Programs
4. A Logic for the Specification of Properties
5. The Model Checking Algorithm
6. A Prototypical Tool
7. Application
8. Concluding Remarks
The Model Checking Algorithm

- Classical tableau algorithm for the LTL model checking problem [CGP99, MP95, LP85].
- To prove that a property is satisfied, it is possible to prove that there is no path in the model checking graph satisfying the negation of the formula.
The Model Checking Algorithm

**Definition (Strongly Connected Component)**

Given a graph $G$, we define a Strongly Connected Component (SCC) $S$ as a maximal subgraph of $G$ such that for every two distinct nodes $A, B \in S$, there exists a path from $A$ to $B$ that passes through nodes of $S$. We say that $S$ is **transient** if it consists of a single node that is not connected to itself.

**Definition (Self-fulfilling SCC)**

Given a model-checking graph $G$, a self-fulfilling strongly connected component $C$ is defined as a non-transient strongly connected component in $G$ which satisfies that for every node $n$ in $C$ and for every $\diamond \phi \in Q_n$ there exists a node $m$ in $C$ such that $\phi \in Q_m$. 

Algorithm:
1. To construct the model checking graph using the negation of $\phi$ (i.e., $\neg\phi$).
The Model Checking Algorithm

**Algorithm:**

1. To construct the model checking graph using the negation of $\phi$ (i.e. $\neg \phi$).
   - First of all, we have to compute the closure $CL(\neg \phi)$. This closure is reminiscent to the Fisher Ladner’s one [MP95, FL79].
The Model Checking Algorithm

Algorithm:

1. To construct the model checking graph using the negation of \( \phi \) (i.e. \( \neg \phi \)).

   - First of all, we have to compute the closure \( CL(\neg \phi) \). This closure is reminiscent to the Fisher Ladner’s one [MP95, FL79].

2. To look for a sequence such that starting from an initial node of the graph that satisfies the negation of \( \phi \), it reaches a self-fulfilling strongly connected component.
The Model Checking Algorithm

Algorithm:

1. To construct the model checking graph using the negation of $\phi$ (i.e., $\neg\phi$).

   - First of all, we have to compute the closure $CL(\neg\phi)$. This closure is reminiscent to the Fisher Ladner’s one [MP95, FL79].

2. To look for a sequence such that starting from an initial node of the graph that satisfies the negation of $\phi$, it reaches a self-fulfilling strongly connected component.

3. If we find a self-fulfilling SCC in the graph, then the system satisfies the property represented by the negated formula (i.e., the model does not satisfy the property).
Outline

1. The tcc Calculus
2. The Model Checking Technique
3. A Model for tcc Programs
4. A Logic for the Specification of Properties
5. The Model Checking Algorithm
6. A Prototypical Tool
7. Application
8. Concluding Remarks
A Prototypical Tool

Property

- We represent a temporal formula as a **binary syntax tree**.

\[ \Diamond \left( (\text{in} = \text{true}) \land \neg \circ (x = 1) \right) \]

\[
>>> \text{phi} = \text{Formula}\left(\{"<>": \{"\neg": \{"": \text{in=true}"\},"\": \{"o": \text{x=1}"\}\}\}\right)
\]
A Prototypical Tool

Model

(a) tcc Structure

Figure: Model of the system
A Prototypical Tool

Model

- We assume that a tcc Structure is a list of tcc nodes.

```
>>> {"store": [Formula({":":"in=true"})], "internal": [], "temporal": ["tell14","p9"], "edges": [2,3], "initial": True}
```
A Prototypical Tool
Model Checking Algorithm

Closure

```python
>>> getClosure(phi, closure)
{'<>': {'~': {'': 'in=true', '~': {'o': 'x=1'}}}
{('~': {'<>' : {'~': {'': 'in=true', '~': {'o': 'x=1'}}}}
{'<o': {'<>' : {'~': {'': 'in=true', '~': {'o': 'x=1'}}}}
...
```

Model Checking Nodes

```python
>>> atoms = getAllAtoms(closure)
>>> model_checking_atoms = getModelCheckingAtoms(tcc_structure, atoms)
tcc State 1
Atom 1
{o': {'<>' : {'~': {'': 'in=true', '~': {'o': 'x=1'}}}}
{': 'in=true'}
...
```
A Prototypical Tool
Model Checking Algorithm

Model Checking Graph

```python
>>> model_checking_graph = getModelCheckingGraph(tcc_structure, model_checking_atoms)
>>> model_checking_graph
{1: [], 2: [], 3: [9, 11, 12, 13, 15], 4: [10, 16, 14], 5: [], 6: [], 7: [9, 11, 12, 13, 15], 8: [10, 16, 14], 9: [], 10: [], 11: [9, 11, 12, 13, 15], 12: [10, 16, 14], 13: [25, 27, 28, 29, 31], 14: [26, 32, 30], 15: [], 16: [], 17: [25, 27, 28, 29, 31], 18: [26, 32, 30], 19: [], 20: [], 21: [25, 27, 28, 29, 31], 22: [26, 32, 30], 23: [], 24: [], 25: [], 26: [], 27: [9, 11, 12, 13, 15], 28: [10, 16, 14], 29: [25, 27, 28, 29, 31], 30: [26, 32, 30], 31: [], 32: []}
```
A Prototypical Tool
Model Checking Algorithm

Strongly Connected Components

```python
>>> strongly_connected_components = tarjan(model_checking_graph)
>>> sccGraphs = getModelCheckingSCCSubgraphs(
    strongly_connected_components, tcc_structure, model_checking_atoms, model_checking_graph)
>>> sccGraphs[0]

[3: [11, 13], 7: [11, 13], 11: [11, 13], 13: [27, 29], 17: [27, 29],
  21: [27, 29], 27: [11, 13], 29: [27, 29]]
```
A Prototypical Tool

Model Checking Algorithm

Self-fulfilling SCC

```python
>>> initialNodes = getInitialNodes(tcc_structure, model_checking_atoms)
>>> initialNodesEntailFormula(sccGraphs[0], initialNodes, model_checking_atoms, formula)
True
>>> isSelfFulfilling(sccGraphs[0], initialNodes, model_checking_atoms)
True
```

Model Checking Algorithm

```python
>>> result = modelSatisfiesProperty(phi, tcc_structure)
is Self Fulfilling: True
Initial Nodes Entail Formula: True
>>> print "Model Satisfies Original Formula: ", not result
Model Satisfies Formula: False
```
A Prototypical Tool

Results

1. tcc Program: See

2. tcc Structure: See

3. Property:
   - $\phi = \square((\text{in} = \text{true}) \Rightarrow \Diamond (x = 1))$
   - $\neg \phi = \Diamond((\text{in} = \text{true}) \land \neg \Diamond (x = 1))$

4. Model Checking Algorithm\textsuperscript{1}:
   - Self-fulfilling SCC: True
   - Initial Nodes Entail Formula: True
   - Model Satisfies Formula: True

5. Conclusion: Model does not satisfy the original formula.

\textsuperscript{1}http://escher.puj.edu.co/~jearias/files/tccModelChecking.zip
Outline

1. The tcc Calculus
2. The Model Checking Technique
3. A Model for tcc Programs
4. A Logic for the Specification of Properties
5. The Model Checking Algorithm
6. A Prototypical Tool
7. Application
8. Concluding Remarks
Model checking is a principal technique for verifying finite-state reactive systems, and in particular, digital systems [Kot08a] (e.g. Traffic light controller [BCDM86, Swa97], radar jammer [CEP03], sequence detector [KK09], binary comparator [Kot08b]).

The coffee machine provides the user with a cup of coffee or a cup of hot water (for tea), given that the consumer has inserted the required amount of money.
Application
Modeling and Verification of Digital Systems

- To describe the digital system using \texttt{tcc}.
- To create the model (\texttt{tcc Structure}) of the system.
- To specify the properties to be verified.
- Use the model checking algorithm to determine whether the model satisfies the property.
Outline

1. The tcc Calculus
2. The Model Checking Technique
3. A Model for tcc Programs
4. A Logic for the Specification of Properties
5. The Model Checking Algorithm
6. A Prototypical Tool
7. Application
8. Concluding Remarks
Concluding Remarks

Overview

- We proposed a model checking algorithm for tcc, an automated formal verification tool.
- Model of the system:
  - Structure (tcc Structure) to model the behavior of a tcc system.
  - Algorithm to construct the tcc Structure from the tcc specification.
  - Method to simplify the model.
- Temporal logic to specify properties of tcc systems.
- Algorithm to determine whether the model of the system satisfies a formula.
- Prototype of the model checking algorithm.
Concluding Remarks

Future Work

• Local operator in the model construction algorithm.
• Automatic tool for constructing the model of the system.
• Symbolic and abstract techniques for reducing the state explosion problem.
• \texttt{ntcc} model checking.
• Prototype improvement (e.g. compute automatically the negated form of the formulas).
Thank you!
References


References

Moreno Falaschi and Alicia Villanueva. 
Automatic Verification of Timed Concurrent Constraint Programs. 

Hubert Garavel. 

C. A. R. Hoare. 
*Communicating Sequential Processes*. 
References


References

Orna Lichtenstein and Amir Pnueli.
Checking that Finite State Concurrent Programs Satisfy their Linear Specification.

Robin Milner.
Communication and Concurrency.

Robin Milner.
Communicating and Mobile Systems: the Pi-Calculus.
References

Zohar Manna and Amir Pnueli.  
*Temporal Verification of Reactive Systems: Safety.*  

Vijay A. Saraswat.  
*Concurrent Constraint Programming.*  

Vijay A. Saraswat, Radha Jagadeesan, and Vineet Gupta.  
Programming in Timed Concurrent Constraint Languages.  

Davide Sangiorgi and David Walker.  
*The Pi-Calculus: A Theory of Mobile Processes.*  
The tcc Calculus [SJG94]

Formal Syntax

(Agents) \[ A ::= c \quad \text{(Tell)} \]
\[ \text{now } c \text{ then } A \quad \text{(Timed Positive Ask)} \]
\[ \text{now } c \text{ else } A \quad \text{(Timed Negative Ask)} \]
\[ \text{next } A \quad \text{(Unit Delay)} \]
\[ \text{abort} \quad \text{(Abort)} \]
\[ \text{skip} \quad \text{(Skip)} \]
\[ A \parallel A \quad \text{(Parallel Composition)} \]
\[ \exists x(A) \quad \text{(Hiding)} \]
\[ g \quad \text{(Procedure Call)} \]

(Procedure Calls) \[ g ::= p(t_1, \ldots, t_n) \]
(Declarations) \[ D ::= g :: A \quad \text{(Definition)} \]
\[ D.D \quad \text{(Conjunction)} \]

(Programs) \[ P ::= D.A \]
The tcc Calculus [SJG94]
Formal Operational Semantics

Axioms for $\rightarrow$. The binary relation $\rightarrow$ on configurations is the least relation satisfying the rules:

- $(?, \text{skip}) \rightarrow ?$
- $(?, \text{abort}) \rightarrow \text{abort}$
- $(?, A \| B) \rightarrow (?, A, B)$
- $(?, \exists x(A)) \rightarrow (?, A[y/x])$ (y not free in ?)
- $(?, p(t_1, \ldots, t_n)) \rightarrow (?, A[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n])$

$\sigma(?) \vdash c$

\[
\frac{}{(?, \text{now } c \text{ then } A) \rightarrow (?, A)}
\]

$\sigma(?) \vdash c$

\[
\frac{}{(?, \text{now } c \text{ else } B) \rightarrow ?}
\]

Axioms for $\Rightarrow$. The binary relation $\Rightarrow$ is the least relation satisfying the single rule:

\[
\Delta, \{\text{now } c_i \text{ else } A_i \mid i < n\} \Rightarrow \\
\Delta, \{\text{now } c_i \text{ else } A_i \mid i < n\}, \{\text{next } B_j \mid j < m\} \Rightarrow \{A_i \mid i < n\}, \{B_j \mid j < m\}
\]
A Model for tcc Programs

Program Labeling

Let $P$ be a tcc specification, the labeled version $P_l$ of $P$ is defined as follows. The subindex $k \in \mathbb{N}$ corresponds to the number of labels introduced up to a given point. When the labeling process starts, $k = 0$ and each time that we introduce a new label, $k$ is incremented by one.

- If $P = \text{tell } c$ then $P_l = \{l_{\text{tell}_k}\}$ $\text{tell } c$.
- If $P = \text{now } c$ then $A$ then $P_l = \{l_{\text{nowp}_k}\}$ $\text{now } c$ then $A_l$.
- If $P = \text{now } c$ else $A$ then $P_l = \{l_{\text{nown}_k}\}$ $\text{now } c$ else $A_l$.
- If $P = \text{next } A$ then $P_l = \{l_{\text{next}_k}\}$ $\text{next } A_l$.
- If $P = \text{skip}$ then $P_l = \{l_{\text{skip}_k}\}$ $\text{skip}$.
- If $P = A \parallel B$ then $P_l = \{l_{||_k}\}$ $(A_l \parallel B_l)$.
- If $P = p(t_1, \ldots, t_n)$ then $P_l = \{l_{p_k}\}$ $p(t_1, \ldots, t_n)$.
A Model for tcc Programs

The tcc Structure

### Definition (tcc State)

Let $AP$ be the atomic propositions in the tcc syntax and $L$ be the set of all labels generated during the labeling process described above. We define the set of states as $S \subseteq 2^{AP} \times 2^{L}$.

### Definition (Equivalent States)

Given two tcc states $s$ and $s'$, we say that two states are equivalent if

1. the set of labels of $s$ and $s'$ coincide, and
2. there exists a renaming $\gamma$ of variables of the constraints in $s$ which makes them syntactically identical to the set of constraints of $s'$. 
A Model for tcc Programs

Construction of the Model

Our procedure consists in locating an active label and perform the actions associated with such agent. The process is performed while there are active labels. When we reach a state where there is no active labels (quiescent point) we have to pass to the next time unit, and then we continue with the procedure. We represent this passage in our graph as follows:

1. Introduce a new node $s'$ related with $s$ by a temporal transition. The state $s$ is a state where there is no active labels.
2. We introduce the temporal labels of $s$ in the labels of $s'$.
3. The store and the temporal labels of $s'$ are empty.
A Model for tcc Programs

Construction of the Model

Tell \( S \equiv \{t_{\text{tell}}\} \text{tell } c \). The new information \( c \) is added to the store in the current time. We translate this behavior to our graph structure as follows:

1. Add a new node \( s' \) related with the node \( s \) from which the agent is execute.
2. The store of \( s' \) is defined as the store of \( s \) plus the constraint \( c \) (i.e. \( C(s') = C(s) \land c \)).
3. The internal labels of \( s' \) are obtained from those of \( s \) by removing \( \{t_{\text{tell}}\} \) (i.e. \( L(s') = L(s) \setminus \{t_{\text{tell}}\} \)).
4. The temporal labels of \( s' \) are the same as \( s \) (i.e. \( LT(s') = LT(s) \)).
Parallel $S \equiv \{l \parallel k\} (A_l \parallel B_l)$. The agents $A$ and $B$ are executed in parallel. We translate this behavior to our graph structure as follows:

1. Introduce a new node $s'$ related with the node $s$ from which the agent is execute.
2. The internal labels of $s'$ are obtained from those of $s$ by adding $A_l$ and $B_l$, and removing $\{l \parallel k\}$. Notice that this corresponds to a concurrent semantics rather than an interleaving interpretation of the parallel operator.
3. The store and the temporal labels of $s'$ are the same as $s$. 


A Model for \texttt{tcc} Programs

Construction of the Model

Timed Positive Ask \( S \equiv \{l_{\text{nowp}_k}\} \text{ now } c \text{ then } A_l \). If the current store entails \( c \) then the agent \( A \) is executed or does nothing otherwise.

Next we describe how to translate this agent to our graph.

1. Add two new nodes \( s'_1, s'_2 \) related with the node \( s \) from which the agent is executed. This branch corresponds to the two possible behaviors.
2. The store of \( s'_1 \) is defined as the union of the store of \( s \) and the constraint \( c \).
3. The store of \( s'_2 \) is defined as the store of \( s \) plus the absence of the constraint \( c \) (i.e. \( C(s'_2) = C(s) \land \neg c \)).
4. The internal labels of \( s'_1 \) are obtained from those of \( s \) by adding \( A_l \) and removing \( \{l_{\text{nowp}_k}\} \).
5. The internal labels of \( s'_2 \) are the same as \( s \).
6. The temporal labels of \( s'_1 \) and \( s'_2 \) are the same as \( s \).
A Model for tcc Programs

Construction of the Model

**Skip** \( S \equiv \{l_{\text{skip}_k}\} \text{skip} \). This agent does nothing at every time instant. We translate this agent to our graph as follows:

1. Construct a new node \( s' \) related with the node \( s \) from which the agent is executed.
2. The internal labels of \( s' \) are obtained from those of \( s \) by removing \( \{l_{\text{skip}_k}\} \).
3. The store and the temporal labels of \( s' \) are the same as \( s \).
A Model for tcc Programs

Construction of the Model

Procedure Call \( S \equiv \{ l_{p_k} \} p(t_1, \ldots, t_n) \). This operator refers to another procedure which have different labels and variables. To translate this agent to our graph we create a new node and the label associated to the first agent of \( p \) is added to the internal labels of the node.
A Model for tcc Programs

Construction of the Model

Timed Negative Ask \( S \equiv \{ l_{\text{nown}_k} \} \text{now } c \text{ else } A_l \). If on the quiescent point the store does not entail \( c \) then the agent \( A \) is executed in the next time instant, or does nothing otherwise. We translate this behavior to our graph as follows:

1. Introduce two new nodes \( s'_1, s'_2 \) related with the node \( s \) from which the agent is execute. This branch corresponds to the two possible behaviors of the agent.
2. The store of \( s'_1 \) is defined as the union of the store of \( s \) and the constraint \( c \).
3. The store of \( s'_2 \) is defined as the store of \( s \) plus the absence of the constraint \( c \).
4. The internal labels of \( s'_1 \) and \( s'_2 \) are the same as \( s \) by removing \( \{ l_{\text{nown}_k} \} \).
5. The temporal labels of \( s'_1 \) are the same as \( s \).
6. The temporal labels of \( s'_2 \) are obtained from those of \( s \) by adding \( A_l \).
Unit Delay $S \equiv \{l_{\text{next}_k}\} \text{next } A_l$. The agent $A$ is executed in the next time instant. We translate this agent to our graph as follows:

1. Construct a new node $s'$ related with the node $s$ from which the agent is execute.
2. The internal labels of $s'$ are the same as $s$ but removing $\{l_{\text{next}_k}\}$.
3. The temporal labels of $s'$ are obtained from those of $s$ by adding $A_l$.
4. The store of $s'$ is the same as $s$. 

A Logic for the Specification of Properties
The ntcc Logic

Definition (CLTL Semantics)

We say that \( \alpha \in C^\omega \) satisfies (or that it is a model of) \( F \) in CLTL, written \( \alpha \models_{\text{CLTL}} F \), iff \( \langle \alpha, 1 \rangle \models_{\text{CLTL}} F \), where:

\[
\begin{align*}
\langle \alpha, i \rangle &\models_{\text{CLTL}} \text{true} \\
\langle \alpha, i \rangle &\models_{\text{CLTL}} \text{false} \\
\langle \alpha, i \rangle &\models_{\text{CLTL}} c \quad \text{iff} \quad \alpha(i) \models c \\
\langle \alpha, i \rangle &\models_{\text{CLTL}} \neg F \quad \text{iff} \quad \langle \alpha, i \rangle \not\models_{\text{CLTL}} F \\
\langle \alpha, i \rangle &\models_{\text{CLTL}} F \land G \quad \text{iff} \quad \langle \alpha, i \rangle \models_{\text{CLTL}} F \quad \text{and} \quad \langle \alpha, i \rangle \models_{\text{CLTL}} G \\
\langle \alpha, i \rangle &\models_{\text{CLTL}} F \lor G \quad \text{iff} \quad \langle \alpha, i \rangle \models_{\text{CLTL}} F \quad \text{or} \quad \langle \alpha, i \rangle \models_{\text{CLTL}} G \\
\langle \alpha, i \rangle &\models_{\text{CLTL}} F \circ \neg F \quad \text{iff} \quad \langle \alpha, i + 1 \rangle \models_{\text{CLTL}} F \\
\langle \alpha, i \rangle &\models_{\text{CLTL}} \Box F \quad \text{iff} \quad \text{for all } j \geq i \quad \langle \alpha, j \rangle \models_{\text{CLTL}} F \\
\langle \alpha, i \rangle &\models_{\text{CLTL}} \Diamond F \quad \text{iff} \quad \text{there is a } j \geq i \quad \text{s.t.} \quad \langle \alpha, j \rangle \models_{\text{CLTL}} F \\
\langle \alpha, i \rangle &\models_{\text{CLTL}} \exists_x F \quad \text{iff} \quad \text{there is an } x\text{-variant } \alpha' \text{ of } \alpha \text{ s.t.} \quad \langle \alpha', i \rangle \models_{\text{CLTL}} F
\end{align*}
\]

Define \( \llbracket F \rrbracket = \{ \alpha \mid \alpha \models_{\text{CLTL}} F \} \). We say that \( F \) is CLTL valid iff \( \llbracket F \rrbracket = C^\omega \), and that \( F \) is CLTL satisfiable iff \( \llbracket F \rrbracket \neq \emptyset \).
The Model Checking Algorithm

Closure

The closure of a formula \( \varphi \), \( CL(\varphi) \), is the smallest set of formulas satisfying the following conditions:

1. \( \varphi \in CL(\varphi) \),
2. \( \neg \varphi_1 \in CL(\varphi) \), iff \( \varphi_1 \in CL(\varphi) \),
3. if \( \varphi_1 \land \varphi_2 \in CL(\varphi) \), then \( \varphi_1, \varphi_2 \in CL(\varphi) \),
4. if \( \varphi_1 \lor \varphi_2 \in CL(\varphi) \), then \( \varphi_1, \varphi_2 \in CL(\varphi) \),
5. if \( \exists x \varphi_1 \in CL(\varphi) \), then \( \varphi_1 \in CL(\varphi) \),
6. if \( \circ \varphi_1 \in CL(\varphi) \), then \( \varphi_1 \in CL(\varphi) \),
7. if \( \neg \circ \varphi_1 \in CL(\varphi) \), then \( \circ \neg \varphi_1 \in CL(\varphi) \),
8. if \( \Box \varphi_1 \in CL(\varphi) \), then \( \varphi_1, \circ \Box \varphi_1 \in CL(\varphi) \),
9. if \( \Diamond \varphi_1 \in CL(\varphi) \), then \( \varphi_1, \circ \Diamond \varphi_1 \in CL(\varphi) \)
The Model Checking Algorithm

Model Checking Graph

Definition (Model-Checking Graph)

Let $\varphi$ be a formula, $CL(\varphi)$ be the closure of $\varphi$ and $Z$ the tcc Structure. A node $n$ of the model-checking graph is formed by a pair of the form $(s_n, Q_n)$ where $s_n$ is a state of $Z$ and $Q_n$ is a subset of $CL(\varphi)$ and the atomic propositions such that the following conditions are satisfied:

1. for every atomic proposition $p$, $p \in Q_n$ iff $p \in C(s_n)$,
2. for every $\exists x \varphi_1 \in CL(\varphi)$, $\exists x \varphi_1 \in Q_n$ iff $\exists x \varphi_1 \in C(s_n)$,
3. for every $\varphi_1 \in CL(\varphi)$, $\varphi_1 \in Q_n$ iff $\neg \varphi_1 \notin Q_n$,
4. for every $\varphi_1 \land \varphi_2 \in CL(\varphi)$, $\varphi_1 \land \varphi_2 \in Q_n$ iff $\varphi_1 \in Q_n$ and $\varphi_2 \in Q_n$,
5. for every $\varphi_1 \lor \varphi_2 \in CL(\varphi)$, $\varphi_1 \lor \varphi_2 \in Q_n$ iff $\varphi_1 \in Q_n$ or $\varphi_2 \in Q_n$,
6. for every $\neg \circ \varphi_1 \in CL(\varphi)$, $\neg \circ \varphi_1 \in Q_n$ iff $\circ \neg \varphi_1 \in Q_n$,
7. for every $\square \varphi_1 \in CL(\varphi)$, $\square \varphi_1 \in Q_n$ iff $\varphi_1 \in Q_n$ and $\circ \square \varphi_1 \in Q_n$,
8. for every $\Diamond \varphi_1 \in CL(\varphi)$, $\Diamond \varphi_1 \in Q_n$ iff $\varphi_1 \in Q_n$ or $\circ \Diamond \varphi_1 \in Q_n$.

An edge in the graph is defined as follows: there will be an edge from one node $n_1 = (s_1, Q_1)$ to another node $n_2 = (s_2, Q_2)$ iff there is an arc from the node $s_1$ to the node $s_2$ in $Z$ and for every formula $\circ \varphi_1 \in CL(\varphi)$, $\circ \varphi_1 \in Q_1$ iff $\varphi_1 \in Q_2$. This means that the next state $s_2$ must satisfy $\phi$ if $s_1$ satisfies $\circ \phi$. 
A Prototypical Tool

Closure

\[
CL(\neg \varphi) = \{ \Box((in = true) \land \neg \circ (x = 1)), \\
\neg \Box((in = true) \land \neg \circ (x = 1)), \\
\circ \Box((in = true) \land \neg \circ (x = 1)), \\
\neg \circ \Box((in = true) \land \neg \circ (x = 1)), \\
\circ \neg \Box((in = true) \land \neg \circ (x = 1)), \\
\circ \neg (in = true) \land \neg \circ (x = 1), \\
\neg ((in = true) \land \neg \circ (x = 1)), \\
\circ (x = 1), \circ (in = true), \\
\circ (x = 1), (x = 1), \neg (x = 1), \\
(in = true), \neg (in = true) \}
\]
A Prototypical Tool

Model Checking Graph
A Prototypical Tool
Strongly Connected Components