Modeling and Verification of Real-time/Hybrid/Cyber-Physical Systems via Concurrent Co-inductive Constraint Logic Programming

Neda Saeedloei

Department of Computer Science
University of Texas at Dallas

May 24th 2011

Outline

Outline

1. Motivation
   - Incorporation of Real Time in Computation
   - Related Work
     - Temporal Logics
     - RTCTL
2. Background
3. Contribution
   - Co-inductive CLP(R) Framework for Verifying Real-time Systems
   - Timed Grammars
     - Practical Parser
   - Timed π-calculus
     - Operational Semantics in LP
   - Foundations of Cyber-Physical Systems (CPS)
4. Summary

Incorporation of Real Time in Computation

Complex real-time systems are difficult to model and verify because they involve:

- Continuous time
- Perpetual execution
- Concurrency

Goal

- Developing techniques for modeling continuous time in real-time systems
  - Co-inductive logic programming
  - Constraint logic programming over reals (CLP(R))

Outline

1. Motivation
   - Incorporation of Real Time in Computation
   - Related Work
     - Temporal Logics
     - RTCTL
2. Background
3. Contribution
   - Co-inductive CLP(R) Framework for Verifying Real-time Systems
   - Timed Grammars
     - Practical Parser
   - Timed π-calculus
     - Operational Semantics in LP
   - Foundations of Cyber-Physical Systems (CPS)
4. Summary

Temporal Logics

- Formalisms for describing sequences of transitions between states in a reactive system
- Can be used for verifying discrete real-time systems
  - Time is not mentioned explicitly
- A powerful example of temporal logics: CTL^*
- Properties like *eventually* or *never* are specified using special temporal operators
- Event *p* will happen within at most *n* time units is not simple to express

Cannot be used in a natural and efficient way to verify many types of interesting properties of real-time systems.

RTCTL

- Obtained by introducing bounds in the CTL temporal operators
- Can be used for verification of discrete real time systems
- Simple and effective way to allow the verification of time bounded properties
- Quantitative analysis on discrete-time models can be performed
  - Computing minimum/maximum delays

Continuous Real-Time

- Time is a continuous quantity
- By discretizing time certain aspects of real-time systems may not be modeled faithfully or at least in a natural fashion
- We model time as a continuous quantity rather than discretizing it
  - Constraint logic programming over reals

ω-Automata

- Nondeterministic finite state automata
- Acceptance condition modified suitably so as to handle infinite input words
- ω-automata accept ω-languages, i.e., a language consisting of infinite words
- A well-known type of ω-automata
  - Büchi automata
    - Some state from the set of final states must be traversed infinitely often

Timed Languages

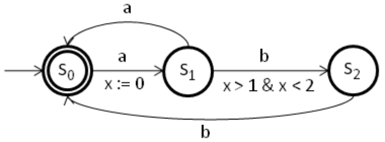
- Behavior of a real-time system can be modeled by a timed word over the alphabet of events
- A timed word over an alphabet $\sum$ is an infinite sequence of pairs of the form $(\sigma_1, \tau_1)(\sigma_2, \tau_2)\ldots$ where
  - σ_i is a symbol from the alphabet $\sum$
  - τ_i is a time-stamp associated with σ_i, such that $\tau_i \in R$ with $\tau_i > 0$ satisfying
    - Monotonicity: τ increases strictly monotonically, that is, $\tau_i < \tau_{i+1}$ for all $i \geq 1$
    - Progress: For every $t \in R$ there is some $i \geq 1$ such that $\tau_i > t$

Timed Automata

- A timed Büchi automaton is a tuple $< \Sigma, S, S_0, C, E, F >$ where
  - Σ is a finite alphabet
  - S is a finite set of states
  - $S_0 \subseteq S$ is a set of start states
  - C is a finite set of clocks
  - $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ gives the set of transitions
  - F is a set of final states

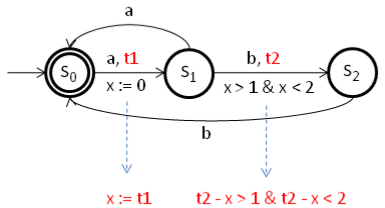
Timed Automata

Example



Timed Automata

Example



Timed Automata are not Enough

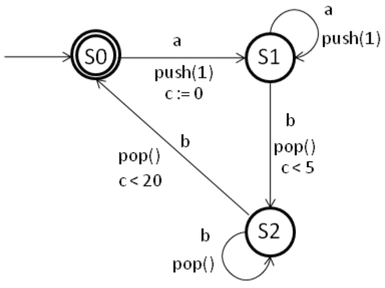
- Using timed automata is a popular approach to designing, specifying and verifying real-time systems
- Equivalent to timed regular ω-languages
- Timed automata are unsuitable for many complex (and useful) applications
- Timed automata are extended to pushdown timed automata

Pushdown Timed Automata (PTA)

- PTA are obtained from timed automata by adding
  - Stack
  - Stack alphabet
  - Stack operations, associated with each transition
- Acceptance conditions for an infinite string for PTA
  - The stack must be empty in every final state

Pushdown Timed Automata

Example



- accepted timed words: $((a, t_a)^n (b, t_b)^n)^\omega$

Outline

Modeling PTA with Co-inductive CLP(R)

- The underlying language is context free, not regular
- Accepted strings are infinite
- Clock constraints model real-time requirements

Framework

Logic programming extended with *co-induction* and *constraints over reals* is used to model PTA

Circularity in Computer Science

- Circular phenomena are quite common in Computer Science:
  - Circular linked lists
  - Graphs (with cycles)
  - Controllers (run forever)
  - Bisimilarity
  - Interactive systems
  - Automata over infinite strings/Kripke structures
  - Perpetual processes
- Numerous other examples can be found elsewhere (Barwise and Moss 1996)

Coinduction

- Infinite structures
  - Some of them can be represented by circular structures
  - Example: $X = [1, 2, 1, 2, ...]$ can be represented by $X = [1, 2 \mid X]$
- Infinite Proofs
  - Exhibit certain regularity such that coinduction can capture them
- Focus of our group: inclusion of coinductive reasoning techniques in LP and its applications

Induction vs Coinduction

- Induction is a mathematical technique for finitely reasoning about an infinite (countable) no. of things.
- Examples of inductive structures:
  - Naturals: 0, 1, 2, ...
  - Lists: [], [X], [X, X], [X, X, X], ...
- Three components of an inductive definition: (1) initiality, (2) iteration, (3) minimality
  - For example, the set of lists is specified as follows:

    An empty list [], is a list (**initiality**) ...(i)
    $[H \mid T]$ is a list if T is a list and H is an element (**iteration**) ...(ii)
    Minimal set that satisfies (i) and (ii) (**minimality**)

Induction vs Coinduction

- Coinduction is a mathematical technique for (finitely) reasoning about infinite things.
- Two components of a coinductive definition: (1) iteration, (2) maximality
  - For example, for a list:

    $[H \mid T]$ is a list if T is a list and H is an element (**iteration**).
    Maximal set that satisfies the specification of a list.
  - This coinductive definition specifies all lists of infinite size.

Mathematical Foundations

Definition	Proof	Mapping
Least fixed point	Induction	Recursion
Greatest fixed point	Coinduction	Corecursion

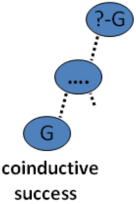
Operational Semantics

p :- p.
The query |?- p. to succeed.

p([1 | T]) :- p(T).
The query |?- p(X) to succeed with X= [1 | X].

Operational Semantics

- Nondeterministic state transition system
- States are pairs of
  - A finite list of syntactic atoms [resolvent] (as in Prolog)
  - A set of syntactic term equations of the form x = f(x) or x = t
- Transition rules
  - Definite clause rule
  - "Coinductive hypothesis rule"
    If a coinductive goal G is called, and G unifies with a call made earlier then G succeeds.



Coinduction

Example: perpetual binary streams

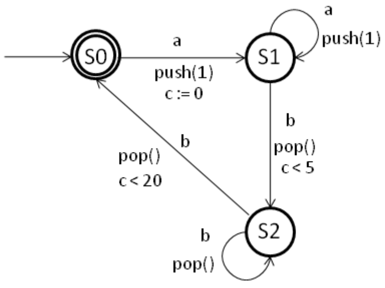
```
bit(0).
bit(1).
bitstream( [ H | T ] ) :- bit( H ), bitstream( T ).
|?- X = [ 0, 1, 1, 0 | X ], bitstream( X ).
```

- Traditional logic program will not terminate.

Example: perpetual binary streams in Coinductive LP

```
:- coinductive stream/1.
stream( [ H | T ] ) :- num( H ), stream( T ).
num( 0 ).
num( s( N ) ) :- num( N ).

|?- stream( [ 0, s( 0 ), s( s ( 0 ) ) | T ] ).
MEMO: stream( [ 0, s( 0 ), s( s ( 0 ) ) | T ] )
MEMO: stream( [ s( 0 ), s( s ( 0 ) ) | T ] )
MEMO: stream( [ s( s ( 0 ) ) | T ] )
      stream(T)
Answers:
T = [ 0, s(0), s(s(0)) | T ]
T = [ s(0), s(s(0)), s(0), s(s(0)) | T ]
T = [ s(s(0)) | T ]    . . .
T = [ 0, s(0), s(s(0)) | X ]  (where X is any
                               rational list of numbers.
```

Example of Modeling PTA with Co-inductive CLP(R)




```
trans(s0, (a, T), s1, Ci, Co, [], [1] ):-{Co=T}.
trans(s1, (a, T), s1, Ci, Co, P, [1|P]):-{Co=Ci}.
trans(s1, (b, T), s2, Ci, Co, [1|P], P):-{T-Ci<5, Co=Ci}
trans(s2, (b, T), s2, Ci, Co, [1|P], P):-{Co=Ci}.
trans(s2, (b, T), s0, Ci, Co, [1|P], P):-{T-Ci<20,Co=Ci}
```

Example of Modeling PTA with Co-inductive CLP(R)

```
:- coinductive(driver/6).
driver([H | R], Si, T, Ci, Pi, [(H, T) | S]) :-
        trans(Si, (H, T), So, Ci, Co, Pi, Po),
        {T2 > T},
        driver(R, So, T2, Co, Po, S).
```

- Input
  - Can be fully specified, e.g., [a,a,a,b,b,b, ...]
  - Can be partially specified, e.g., [a,X,a,Y,b,b, ...]
  - Can be unspecified, e.g., X
- Output
  - Concrete legal behavior of the system
  - Sequences of time-stamped events
    - Time-stamps are not concrete, but related by set of constraints
  - More general than what you normally expect

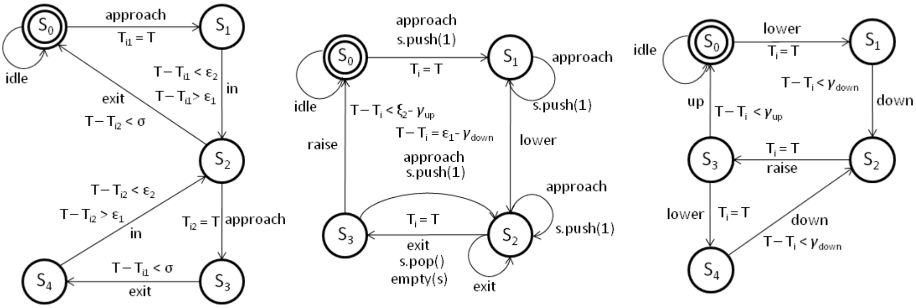
Example of Modeling PTA with Co-inductive CLP(R)

```
[(a,0), (a,2), (b,4), (b,16),...]
         % is legal
         % (will unify with the output of the program)
[(a,0), (a,2), (b,6), (b,16),...]
         % is not legal

[(a,0), (a,2), (b,4), (b,8), (b,16),...]
         % is not legal
```

Application: The Generalized Railroad Crossing (GRC) Problem

- Several tracks and an unspecified number of trains traveling in both directions
- A gate at the railroad crossing, operated (by a controller), in a way that guarantees
  - Safety: The gate must be down while one or more trains are in the crossing
  - Utility: The gate goes down only if a train is approaching

GRC

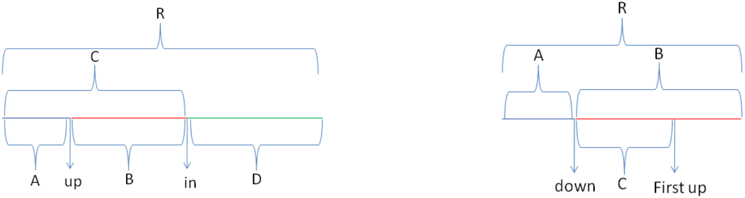

approach
T_i1 = T
idle
T − T_i1 < ε_2
T − T_i1 > ε_1
in
exit
T − T_i2 < σ
T − T_i2 < ε_2
T − T_i2 > ε_1
in
T_i2 = T
approach
T − T_i1 < σ
exit
approach
s.push(1)
T_i = T
approach
s.push(1)
idle
T − T_i < ξ_2 - γ_up
T − T_i = ε_1 - γ_down
raise
lower
approach
s.push(1)
approach
s.push(1)
T_i = T
exit
s.pop()
empty(s)
exit
idle
lower
T_i = T
T − T_i < γ_down
down
up
T − T_i < γ_up
T_i = T
raise
lower
T_i = T
down
T − T_i < γ_down

Verifying Properties

Given a property Q to be verified

- Specify its negation as a logic program, notQ
- If the query notQ fails w.r.t. the logic program that models the system, the property Q holds.
- If the query notQ succeeds, the answer provides a counterexample to why the property Q does not hold.

Verifying Safety and Utility




```
unsafe(N) :- driver( s0, s0, 0, 0, 0, X, N, R ),
             append( C, [ (in(_), _) | D ], R ),
             append( A, [ (up(_), _) | B ], C ),
             not_member(  (down, _), B ).

unutilized(N) :- driver( s0, s0, 0, 0, 0, X, N, R ),
                 append( A, [ (down, _) | B], R ),
                 find_first_up( B, C ),
                 not_member( (in(_), _), C ).
```

Verification Time

Table: safety and utility verification times

Number of tracks	safety	utility
1	0.006	0.006
2	0.065	0.072
3	0.6	0.587
4	5.666	5.634
5	60.013	60.430
6	426.300	453.544

Outline

Motivation

- For real-time systems timed regular languages may not be powerful enough
- Timed context-free languages might be needed

Outline

- We propose timed grammars
  - Simple and natural method for describing timed languages
  - Describe words that have real-time constraints placed on the times at which the word's symbols appear
- Equivalence of PTA and ω-TCFGs
- Modeling ω-TCFGs with
  - Definite clause grammars (DCGs)
  - Constraints over reals (CLP(R))
  - Co-induction

Complex real-time systems can be directly (and naturally) modeled as co-inductive CLP(R) programs

Timed Context-Free Grammars Examples

$$S \rightarrow a \ \{c := 0\} \ \ S$$
$$S \rightarrow b \ \{c < 5\}$$

$$S \rightarrow a \ \{c := 0\} \ \ R$$
$$R \rightarrow a \ \ R$$
$$R \rightarrow b \ \{c < 5\}$$

Timed Context-Free ω-Grammars (ω-TCFGs)

- Timed Context-free grammars with co-recursive grammar rules (i.e., recursive rules that need not have base cases)

Example

$S \rightarrow R \;\; S$
$R \rightarrow a \;\; \{c := 0\} \;\; T \;\; b \;\; \{c < 20\}$
$T \rightarrow a \;\; T \;\; b$
$T \rightarrow a \;\; b \;\; \{c < 5\}$

Modeling Timed Context-Free ω-Grammars with Co-inductive CLP(R)

Incorporation of co-induction and CLP(R) into DCGs allows modeling of ω-TCFGs, this model serves as a practical parser for the ω-TCFL recognized by the ω-TCFG

- General method of Converting ω-TCFGs to co-inductive CLP(R) programs
  - The generated LP models the ω-TCFG as a collection of DCG rules
  - Each rule is extended with clock expressions

Example: Parser

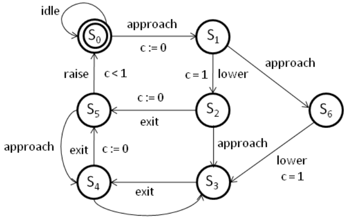
```
s(T, Ci, Co)--> r(T,Ci, Co1), {T2 > T}, s1(T2, Co1, Co).

r(T, Ci, Co)--> [(a, T)], {Ci = T, T2 > T},
                t(T2, Ci, Co1), {T3 > T2},
                [(b, T3)], {T3 - Ci < 20}.

t(T, Ci, Co)--> [(a, T)], {T2 > T}, t(T2, Ci, Co1),
                {T3 > T2}, [(b, T3)], {Co = Co1}.

t(T, Ci, Co)--> [(a, T)], {T2 > T}, [(b, T2)],
                {T2 - Ci < 5, Co = Ci}.
```

Timed Context-Free ω-Grammars Modeled as Co-inductive CLP(R) Programs

- Check whether a particular timed string will be accepted or not
- Systematically generate all possible timed strings that can be accepted
- Verify system properties by posing appropriate queries

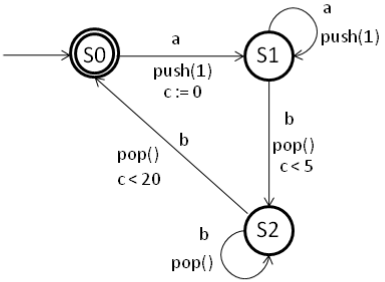
Timed Context-Free ω-Grammar Example

$C \to approach\{c := 0\}\ \ L\ \ exit\{c := 0\}\ \ raise\{c < 1\}\ \ C$
$C \to approach\{c := 0\}\ \ L\ \ N\ \ exit\{c := 0\}\ \ raise\{c < 1\}\ \ C$
$L \to lower\{c < 1\}$
$L \to approach\ \ lower\{c < 1\}\ \ exit$
$N \to approach\ \ exit$
$N \to approach\ \ exit\ \ N$
$N \to exit\ \ approach$
$N \to exit\ \ approach\ \ N$



Equivalence of PTA and ω-CFGs

$$S \rightarrow R \;\; S$$
$$R \rightarrow a \;\; \{c := 0\} \;\; T \;\; b \;\; \{c < 20\}$$
$$T \rightarrow a \;\; T \;\; b$$
$$T \rightarrow a \;\; b \;\; \{c < 5\}$$



Outline

Motivation

- π-calculus was introduced with the aim of modeling concurrent/mobile processes
- It is not equipped to model concurrent real-time systems and reason about their behavior
  - Several extensions of π-calculus with time have been proposed
  - All these approaches discretize time rather than represent it faithfully as a continuous quantity

Outline

- Extending π-calculus with real time by adding clocks
  - Powerful formalism for describing concurrent real-time systems and reasoning about their behaviors
- Developing operational semantics for the proposed timed π-calculus
- Developing the notion of timed bisimilarity and its properties (not presented here)
  - e.g., expansion theorem for real-time, concurrent, mobile processes
- Implementation based on co-induction, coroutining, and constraint logic programming over reals of operational semantics
- Application Example

Design Decisions

- Associating time-stamps to all messages
- Adding clocks
- Adding clock operations
  - Clock resets
  - Clock constraints
- Representing messages by triples of the form $\langle m, t_m, c \rangle$

Syntax

$$C ::= C_c \; C_r$$

$$C_c ::= (Clock \sim x)C_c \mid (Clock - t \sim x)C_c \mid \epsilon$$

$$C_r ::= (Clock := 0)C_r \mid \epsilon$$

$$\sim ::= < \mid > \mid \leq \mid \geq \mid =$$

$$M ::= C\bar{x}\langle y, t_y, c\rangle.P \mid Cx(\langle y, t_y, c\rangle).P \mid C\tau.P \mid 0 \mid M + M'$$

$$P ::= M \mid P \mid P' \mid !P \mid \nu z \; P \mid [x = y] \, P$$

Examples

Example 1

The expression $x(\langle m, t_m, c\rangle).(c - t_m \geq 5)\bar{y}\langle n, t_n, c\rangle$ represents a process that receives a message m on channel x and sends a message n on channel y with the delay of at least 5 units of time.

Example 2

Consider a system which is composed of two processes P and Q that run in parallel. Moreover, there is a clock c that can be accessed by both P and Q which should be reset before the parallel execution begins. The timed π-calculus expression presenting this scenario is $(c := 0)\tau.(P \mid Q)$.

Actions

$$\alpha_t ::= C_r, \bar{x}\langle y, t_y, c\rangle \mid C_r, x(\langle y, t_y, c\rangle) \mid C_r, \bar{x}(\langle y, t_y, c\rangle) \mid C_r, \langle \tau, t\rangle$$

- $P \xrightarrow{C_r,\bar{x}\langle y,t_y,c\rangle} Q$: P sends $\langle y, t_y, c\rangle$ via x, and evolves to Q.
- $P \xrightarrow{C_r,x(\langle y,t_y,c\rangle)} Q$: P receives any message $\langle w, t_w, d\rangle$ and becomes $Q\{w/y, t_w/t_y, d/c\}$.
- $P \xrightarrow{C_r,\bar{x}(\langle y,t_y,c\rangle)} Q$: P emits a private name along with its time-stamp and a clock on port x, and becomes Q.
- $P \xrightarrow{C_r,\langle\tau,t\rangle} Q$: P takes an internal action at time t.
- The set of clocks that should be reset in each transition is specified by C_r.

Timed π-calculus Operational Semantics

$$\text{TAU}\ \frac{[C_c]}{C_c C_r \tau.P \xrightarrow{C_r,\langle \tau, t\rangle} P}$$

$$\text{OUT}\ \frac{[C_c]}{C_c C_r \bar{x}\langle y, t_y, c\rangle.P \xrightarrow{C_r,\ \bar{x}\langle y, t_y, c\rangle} P}$$

$$\text{INP}\ \frac{[C_c\{d/c\}]}{C_c C_r x(\langle z, t_z, c\rangle).P \xrightarrow{C_r\{d/c\}, x(\langle y, t_y, d\rangle)} P\{y/z, t_y/t_z, d/c\}}\ y \notin fn(\nu z P),\ d \notin c(P)$$

$$\text{MAT}\ \frac{P \xrightarrow{\alpha_t} P'}{[x = x]P \xrightarrow{\alpha_t} P'} \qquad \text{SUM}\ \frac{P \xrightarrow{\alpha_t} P'}{P + Q \xrightarrow{\alpha_t} P'}$$

$$\text{PAR}\ \frac{P \xrightarrow{\alpha_t} P'}{P \mid Q \xrightarrow{\alpha_t} P' \mid Q}\ bn(\alpha_t) \cap fn(Q) = \emptyset$$

$$\text{COM}\ \frac{P \xrightarrow{C_r, \bar{x}\langle y, t, c\rangle} P' \quad Q \xrightarrow{C'_r, x(\langle y, t, c\rangle)} Q'}{P \mid Q \xrightarrow{C_r, C'_r, \langle \tau, t\rangle} P' \mid Q'}$$

$$\text{CLOSE}\ \frac{P \xrightarrow{C_r, \bar{x}(\langle z, t, c\rangle)} P' \quad Q \xrightarrow{C'_r, x(\langle z, t, c\rangle)} Q'}{P \mid Q \xrightarrow{C_r, C'_r, \langle \tau, t\rangle} \nu z(P' \mid Q')}\ z \notin fn(Q)$$

$$\text{RES}\ \frac{P \xrightarrow{\alpha_t} P'}{\nu z P \xrightarrow{\alpha_t} \nu z P'}\ z \notin n(\alpha_t)$$

$$\text{OPEN}\ \frac{P \xrightarrow{C_r, \bar{x}\langle y, t_y, c\rangle} P'}{\nu y P \xrightarrow{C_r, \bar{x}(\langle y, t_y, c\rangle)} P'}\ y \neq x$$

$$\text{REP-ACT}\ \frac{P \xrightarrow{\alpha_t} P'}{!P \xrightarrow{\alpha_t} P' \mid\ !P}$$

$$\text{REP-COM}\ \frac{P \xrightarrow{C_r, \bar{x}\langle y, t, c\rangle} P' \quad P \xrightarrow{C'_r, x(\langle y, t, c\rangle)} P''}{!P \xrightarrow{C_r, C'_r, \langle \tau, t\rangle} (P' \mid P'') \mid\ !P}$$

$$\text{REP-CLOSE}\ \frac{P \xrightarrow{C_r, \bar{x}(\langle z, t, c\rangle)} P' \quad P \xrightarrow{C'_r, x(\langle z, t, c\rangle)} P''}{!P \xrightarrow{C_r, C'_r, \langle \tau, t\rangle} (\nu z(P' \mid P'')) \mid\ !P}\ z \notin fn(P)$$

Timed π-calculus Operational Semantics

TAU	$\dfrac{[C_c]}{C_c C_r \tau.P \xrightarrow{C_r, \langle \tau, t \rangle} P}$
OUT	$\dfrac{[C_c]}{C_c C_r \bar{x}\langle y, t_y, c \rangle.P \xrightarrow{C_r,\ \bar{x}\langle y, t_y, c \rangle} P}$
INP	$\dfrac{[C_c\{d/c\}]}{C_c C_r x(\langle z, t_z, c \rangle).P \xrightarrow{C_r\{d/c\}, x(\langle y, t_y, d \rangle)} P\{y/z, t_y/t_z, d/c\}}\ y \notin fn(\nu z P),\ d \notin c(P)$
MAT	$\dfrac{P \xrightarrow{\alpha_t} P'}{[x = x]P \xrightarrow{\alpha_t} P'}$
SUM	$\dfrac{P \xrightarrow{\alpha_t} P'}{P + Q \xrightarrow{\alpha_t} P'}$
PAR	$\dfrac{P \xrightarrow{\alpha_t} P'}{P \mid Q \xrightarrow{\alpha_t} P' \mid Q}\ bn(\alpha_t) \cap fn(Q) = \emptyset$

Table: Timed π-calculus Transition Rules for TAU, OUT, INP, MAT, SUM, PAR

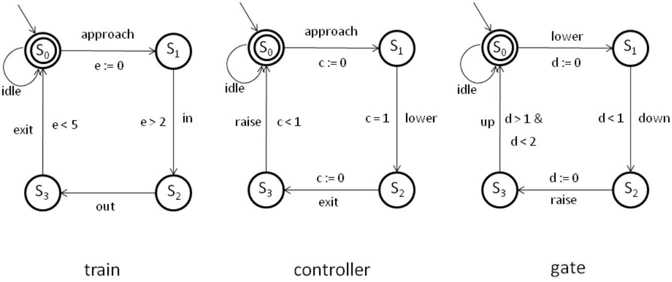
Operational Semantics in Logic Programming

Syntax of the Language in LP

$$\mathcal{A} ::= \mathit{out}((\mathcal{C},\mathcal{N},\mathcal{M}),\mathcal{P}) \mid \mathit{in}((\mathcal{C},\mathcal{N},\mathcal{M}),\mathcal{P}) \mid \mathit{tau}((\mathcal{C},\mathcal{T}),\mathcal{P}) \mid$$
$$\mathit{zero} \mid \mathit{choice}(\mathcal{P},\mathcal{P}) \mid \mathit{par}(\mathcal{P},\mathcal{P}) \mid \mathit{rep}(\mathcal{P}) \mid \mathit{nu}(\mathcal{N},\mathcal{P}) \mid$$
$$\mathit{match}(\mathcal{N} = \mathcal{N},\mathcal{P})$$
$$\mathcal{C} ::= \mathit{reset}(\mathcal{CN}) \mid \mathit{const}(\mathcal{CN} \sim \mathcal{R}) \mid \mathit{const}(\mathcal{CN} - \mathcal{T} \sim \mathcal{R})$$

$$\mathcal{D} ::= \mathit{proc}(\mathcal{PN},\mathcal{P})$$
$$\mathcal{M} ::= (\mathcal{N},\mathcal{T},\mathcal{CN})$$

Example: 1-Track GRC



Example: 1-Track GRC

$$\begin{aligned} train \equiv\ & \nu\, pc\ \overline{ch1}\langle pc, t_p, t\rangle. \\ & (t := 0)\ \overline{pc}\langle approach, t_a, t\rangle. \\ & (t > 2)\ (\tau, t_i). \\ & (\tau, t_o). \\ & (t < 5)\ \overline{pc}\langle exit, t_e, t\rangle) \end{aligned}$$

```
proc(train,
  nu(out(ch1, (pc,tp,t)),
     in(reset(p), pc, (approach,ta,t)),
     tau((t>2)(t<3), ti),
     tau(to),
     out((t<5), pc, (exit,te,t))))
```

Example: 1-Track GRC

$$
\begin{aligned}
\text{controller} \equiv\ & ch1(\langle pc, t, c\rangle).pc(\langle x_1, t_1, c\rangle). \\
& (c = 1)(c := 0)\ \overline{ch2}\langle lower, t_l, c\rangle\ . \\
& pc(\langle x_2, t_2, c\rangle).(c\text{-}t_2 < 1)(c := 0)\ \overline{ch2}\langle raise, t_r, c\rangle
\end{aligned}
$$

```
proc(controller,
  in(ch1, (pc,tp,c)),
  in(pc, (x1,t1,c)),
  out((c=1)(c:=0), ch2, (lower,tl,c)),
  in(pc, (x2,t2,c)),
  out((c<1)(c:=0), ch2, (raise,tr,c)))
```

Example: 1-Track GRC

$$\begin{aligned} gate \equiv\ & ch2(\langle x, t_x, g\rangle). \\ & ([x = lower](g < 1)\ (\tau, t_d) + \\ & \ [x = raise](g > 1)(g < 2)\ (\tau, t_u)) \end{aligned}$$

```
proc(gate,
   in(ch2, (x,tx,g)),
   choice(match(x=lower, tau((g<1), td)),
          match(x=raise, tau((g>1)(g<2), tu))))
```

Example: 1-Track GRC

```
train(X, Y, W, Tc, Si) :-
    (H = approach, {Tc2 = W};
     H = in, {W - Tc > 2, Tc2 = Tc};
     H = out, {Tc2 = Tc};
     H = exit, {W - Tc < 5, Tc2 = Tc}),
     {W2 > W},
     train_trans(Si, H, So),
     freeze(X, train(Xs, Ys, W2, Tc2, So)),
     ((H = approach; H = exit) -> Y = [(H, W)| Ys];
                                  Y = Ys),
     X = [(H, W)| Xs].
```

Example: 1-Track GRC

```
controller([(H, W)| Xs], Y, Sc) :-
   freeze(Xs, controller(Xs, Ys, Sc3)),
   (H = approach, M = lower, {W2 > W, W2 - W = 1};
    H = exit, M = raise, {W2 > W, W2 - W < 1}),
   controller_trans(Sc,  H, Sc2),
   controller_trans(Sc2, M, Sc3),
   Y = [(M, W2)| Ys].

gate([(H, W)| Xs], Sg) :-
   freeze(Xs, gate(Xs, Sg3)),
   (H = lower, M = down, {W2 > W, W2 - W < 1};
    H = raise, M = up, {W2 > W, W2 - W > 1, W2 - W < 2}),
   gate_trans(Sg, H, Sg2), gate_trans(Sg2, M, Sg3).

main(A, B, C) :-
   freeze(A, (freeze(C, gate(C, s0)),
   controller(B, C, s0))), train(A, B, 0, 0, s0).
```

Internal Transitions of GRC Components

```
train-trans(s0, approach, s1).
train-trans(s1, in,       s2).
train-trans(s2, out,      s3).
train-trans(s3, exit,     s0).

c-trans(s0, approach, s1).
c-trans(s1, lower,    s2).
c-trans(s2, exit,     s3).
c-trans(s3, raise,    s0).

g-trans(s0, lower, s1).
g-trans(s1, down,  s2).
g-trans(s2, raise, s3).
g-trans(s3, up,    s0).
```

Outline

Motivation

- CPS consist of perpetually and concurrently executing physical and computational components
- The presence of physical components require the computational components to deal with continuous quantities

CPS Characteristics Summary

- Perform discrete computations
- Deal with continuous physical quantities
- Run forever
- They are concurrent

Design Challenges of CPS

- Dealing with continuous quantities in computations
  - typical approaches discretize them, e.g., time
- Operational modeling/analysis of perpetual computations is not well understood
  - Co-induction have been introduced to formally model rational, infinite computations
- Concurrency is reasonably well understood
- However, concurrency combined with continuous quantities and perpetual computations makes modeling of CPS difficult

Problem

A formalism that can model discrete and continuous quantities together with concurrent, perpetual execution is lacking

Goal

Faithfuly modeling CPS and reasoning about them

Our Thesis

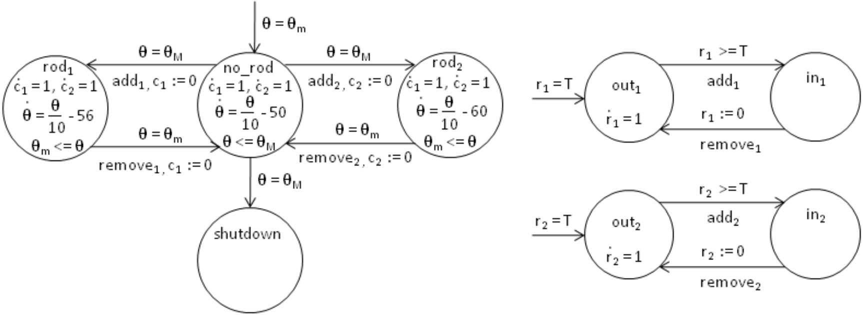
Logic programming extended with *co-induction*, *constraints over reals* and *coroutining* is an excellent formalism for modeling CPS and reasoning about them.

Modeling CPS

- Communicating hybrid ω-automata as underlying model
  - State machines modeled as *logic programs*
  - Physical quantities are represented as continuous quantities (i.e., not discretized)
    - The constraints imposed on them by CPS physical interactions are faithfully modeled with *CLP(R)*
- Non-terminating nature handled via *co-inductive LP*
- The communication/concurrency is handled by *coroutining*

- So each hybrid ω-automaton modeled as a co-inductive CLP(R) program
- The multiple co-inductive CLP(R) programs execute concurrently modeled as co-routined logic programs

Traditional Example of CPS: Reactor Temperature Control System



LP Realization of Reactor Temperature Control System

```
r1(out1,add1,in1,W,Ti,To,T) :-{W-Ti>=T, To=Ti}.
r1(in1,remove1,out1,W,Ti,To,T) :- {To=W}.

r2(out2,add2,in2,W,Ti,To,T) :-{W-Ti>=T, To=Ti}.
r2(in2,remove2,out2,W,Ti,To,T) :- {To=W}.
```

LP Model of Reactor Temperature Control System

```
c(norod,add1,rod1,Pi,Po,W,Ti1,Ti2,To1,To2,F) :-
 (F == 1 -> {Ti=Ti1}; {Ti=Ti2}),
 {Pi<550, Po=550, exp(e,(W-Ti)/10)=5, To1=W, To2=Ti2}.

c(rod1,remove1,norod,Pi,Po,W,Ti1,Ti2,To1,To2,F) :-
 {Pi>510, Po=510, exp(e,(W-Ti1)/10)=5, To1=W, To2=Ti2}.

c(norod,add2,rod2,Pi,Po,W,Ti1,Ti2,To1,To2,F) :-
 (F == 1 -> {Ti=Ti1}; {Ti=Ti2}),
 {Pi<550, Po=550, exp(e,(W-Ti)/10)=5, To1=Ti1, To2=W}.

c(rod2,remove2,norod,Pi,Po,W,Ti1,Ti2,To1,To2,F) :-
 {Pi>510, Po=510, exp(e,(T-Ti2)/10)=9/5, To1=Ti1, To2=W}.

c(norod,_,shutdown,Pi,Po,W,Ti1,Ti2,To1,To2,F) :-
 (F == 1 -> {Ti=Ti1}; {Ti=Ti2}),
 {Pi<550, Po=550, exp(e,(W-Ti)/10)=5, To1=Ti1, To2=Ti2}.
```

LP Model of Reactor Temperature Control System

```
:- coinductive(rod1/6).
rod1([(H, W)| Xs], Si1, Si2, Ti1, Ti2, T) :-
((H = add1; H = remove1) ->
 (H = add1 -> freeze(Xs,rod1(Xs, So1, Si2, To1, Ti2, T));
   freeze(Xs,rod1(Xs, So1, Si2, To1, Ti2, T);
              rod2(Xs, So1, Si2, To1, Ti2, T))),
 r1(Si1, H, So1, W, Ti1, To1, T);
H = shutdown, {W - Ti1 < T, W - Ti2 < T}).

:- coinductive(rod2/6).
rod2([(H, W)| Xs], Si1, Si2, Ti1, Ti2, T) :-
((H = add2; H = remove2) ->
 (H = add2 -> freeze(Xs,rod2(Xs, Si1, So2, Ti1, To2, T));
   freeze(Xs,rod1(Xs, Si1, So2, Ti1, To2, T);
              rod2(Xs, Si1, So2, Ti1, To2, T))),
 r2(Si2, H, So2, W, Ti2, To2, T);
H = shutdown, {W - Ti1 < T, W - Ti2 < T}).
```

LP Model of Reactor Temperature Control System

```
:- coinductive(contr/7).
contr(X, Si, W, Pi, Ti1, Ti2, Fi) :-
(H=add1; H=remove1; H=add2; H=remove2; H=shutdown),
{W2 > W},
freeze(X,contr(Xs, So, W2, Po, To1, To2, Fo)),
c(Si,H,So,Pi,Po,W,Ti1,Ti2,To1,To2,Fi),
((H=add1; H=remove1) -> Fo = 1; Fo = 2),
((H=add1; H=remove1; H=add2; H=remove2) ->
          X = [(H, W)| Xs]; X = [(H, W)]).

main(S, W, T) :-
 {W - Tr1 = T, W - Tr2 = T},
 freeze(S, (rod1(S, s0, s0, Tr1, Tr2, T);
            rod2(S, s0, s0, Tr1, Tr2, T))),
 contr(S, s0, W, 510, Tc1, Tc2, 1).
```

Summary

- Techniques for incorporation of continuous time in computation
  - Co-inductive CLP(R) framework for modeling and verification of real-time systems
  - Timed Grammars
    - Practical parsers
  - Timed π-calculus
    - Operational Semantics in LP
  - Foundations of CPS
- Future work
  - Incorporation of continuous time in traditional model checkers

Publications

Gopal Gupta, **Neda Saeedloei**, Brian DeVries, Richard Min, “Practical Applications of Co-inductive Logic Programming,” To appear in the International Conference on Algebra and Coalgebra (CALCO) 2011.

Neda Saeedloei, Gopal Gupta, “A Logic-based Modeling and Verification of CPS,” To appear in Proceedings of International Conference on Cyber-Physical Systems, Work-in-Progress (WiP) session 2011, SIGBED review.

Neda Saeedloei, Gopal Gupta, “Verifying Complex Continuous Real-Time Systems with Coinductive CLP(R),” Proceedings of the LATA 2010, Springer Verlag, Pages 536-548.

Publications

Neda Saeedloei, Gopal Gupta, "A Logic Programming Realization of Timed Context-Free Grammars," Proceedings of the ICLP 2010, Pages 212-221.

Neda Saeedloei, Gopal Gupta, "Timed π-Calculus," Submitted to TIME 2011.

Neda Saeedloei, Gopal Gupta, "Logic Programming Realization of Timed π-Calculus," In Preparation (to be Submitted to FORMATS 2011).

Neda Saeedloei, Gopal Gupta, "Timed π-Calculus and its applicatins," In Preparation (to be Submitted to the Journal of Science of Computer Programming, Elsevier).

Publications

Neda Saeedloei, Gopal Gupta, “Verifying Complex Continuous Real-Time Systems with Coinductive CLP(R),” Workshop Proceedings of ICLP 2009.

Neda Saeedloei, Gopal Gupta, “Modeling and Verification of Cyber-Physical Systems with Co-inductive Constraint Logic Programming,” In Preparation.