Modeling music processes using temporal concurrent constraint programming

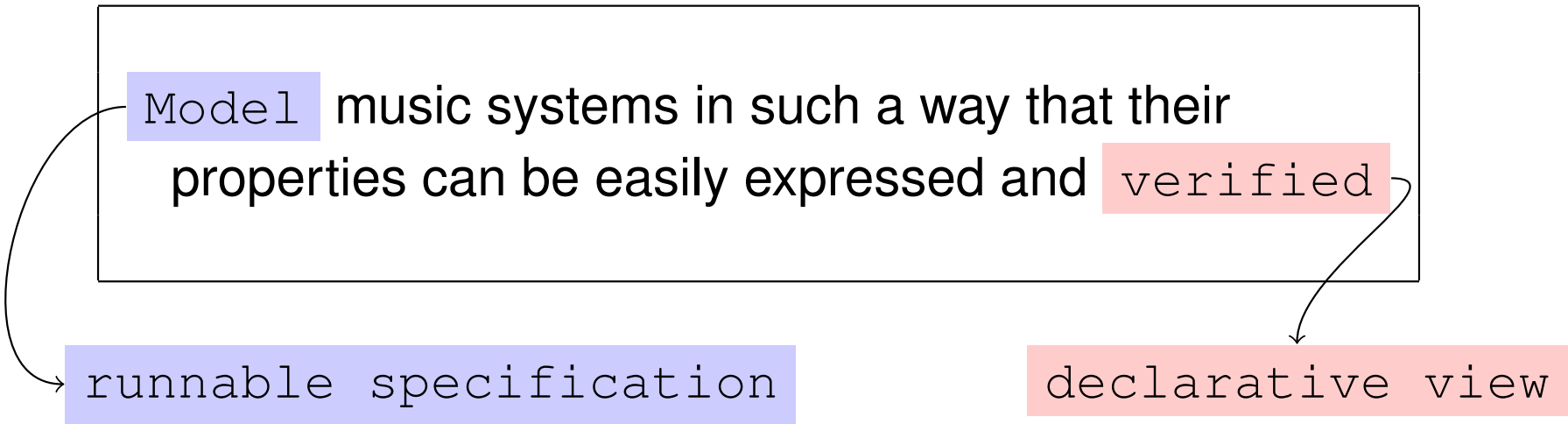
Camilo Rueda
Frank Valencia, Carlos Olarte

Ircam 2011

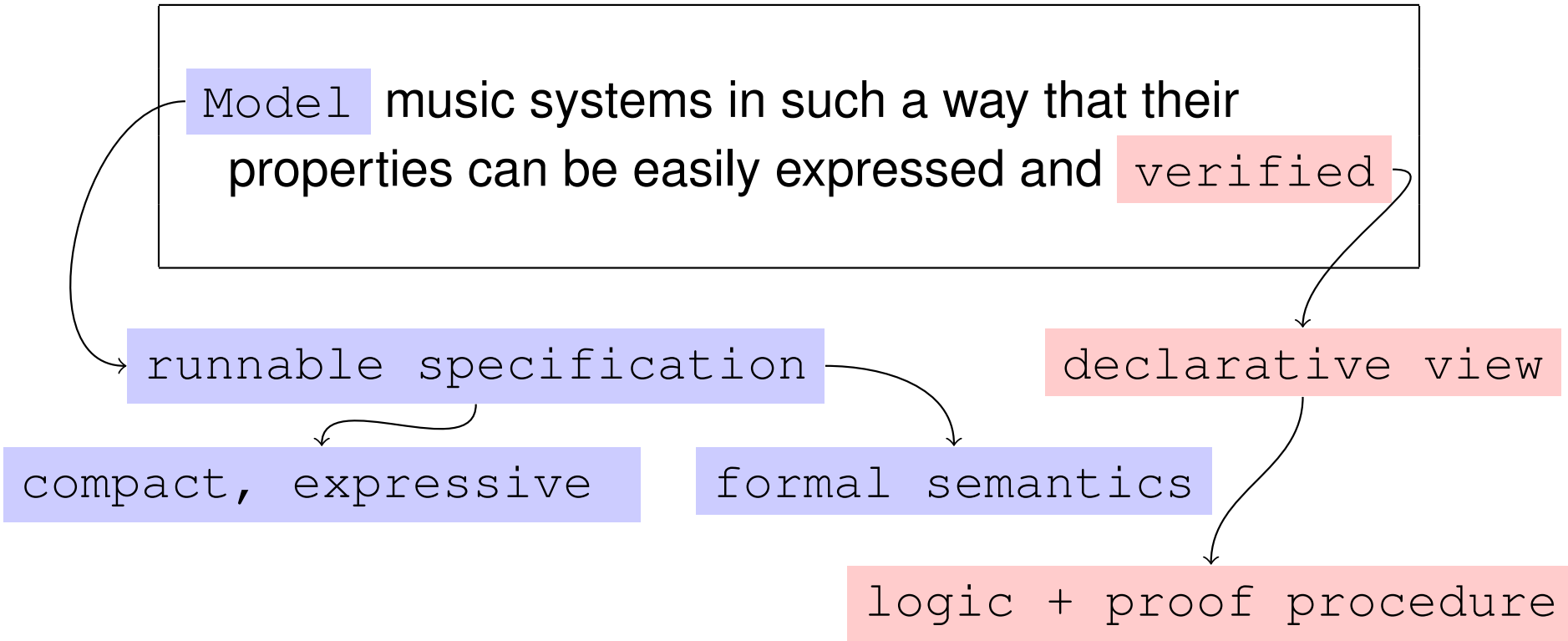
Motivation: the idea behind using ccp

`Model` music systems in such a way that their properties can be easily expressed and `verified`

Motivation: the idea behind using ccp



Motivation: the idea behind using ccp



What types of music systems?

- Reactive: improvisation, interactive performance
- Dynamic: evolving music structures

> Those involving complex synchronization patterns

What constraints are used for

- To communicate partial information
- To synchronize concurrent processes

> The type of constraints (constraint system) is a parameter of the model

Outline

Outline

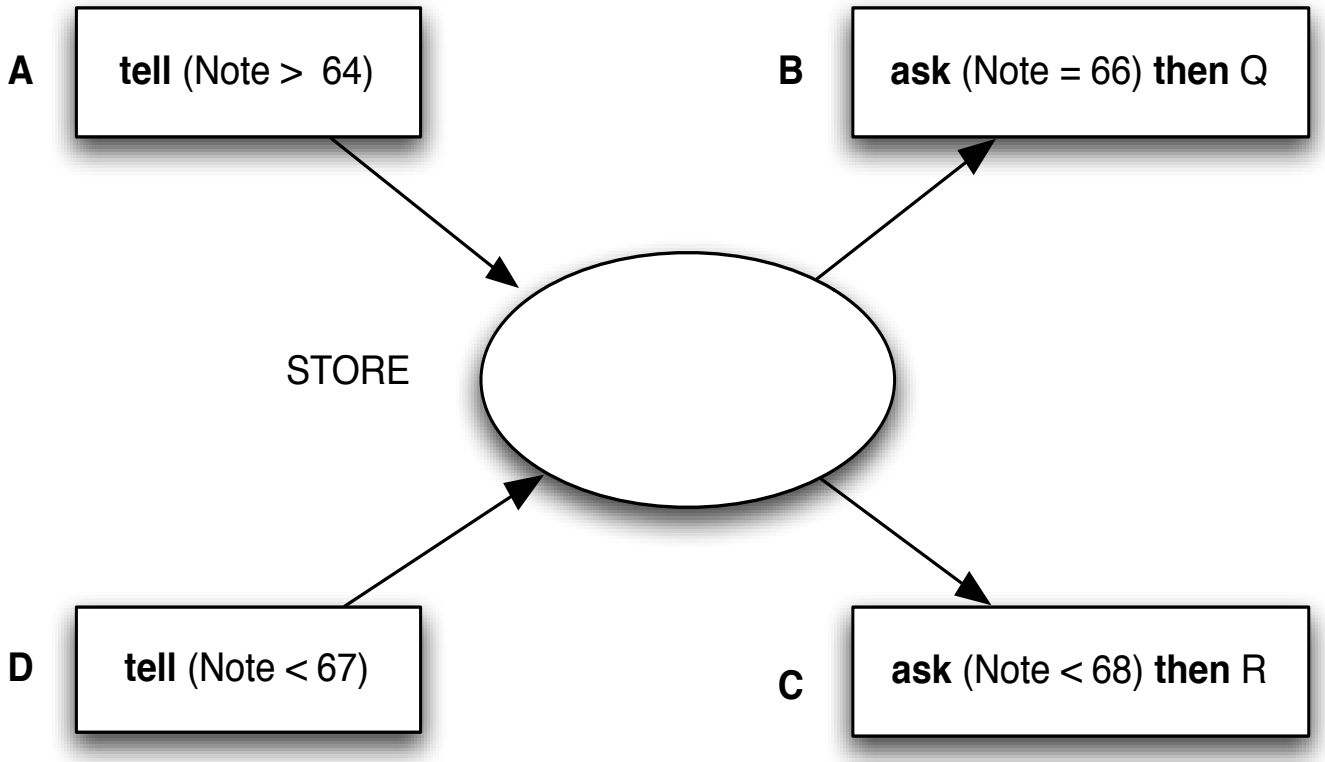
Components of ccp

- A constraint system
  - A set D of tokens
  - An entailment relation, $A \vdash c$

    A constraint is some subset of D (closed by entailment)
- A store of constraints
- Control mechanisms,
  - $\mathbf{tell}(c)$
  - $\mathbf{ask}\ c\ \mathbf{then}\ P$
- Some extra logical operators

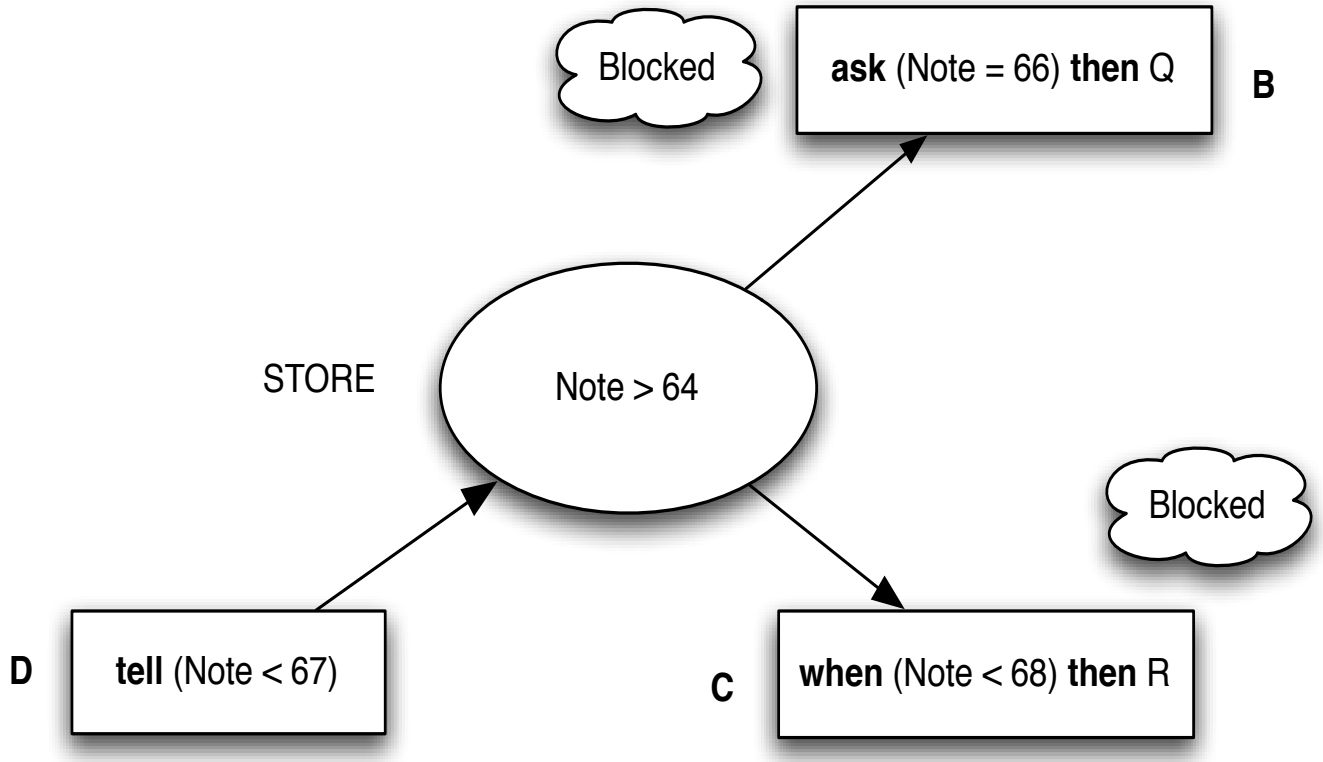
Computation in ccp

ccp's store-as-constraint vs Von Neumann's store-as-valuation



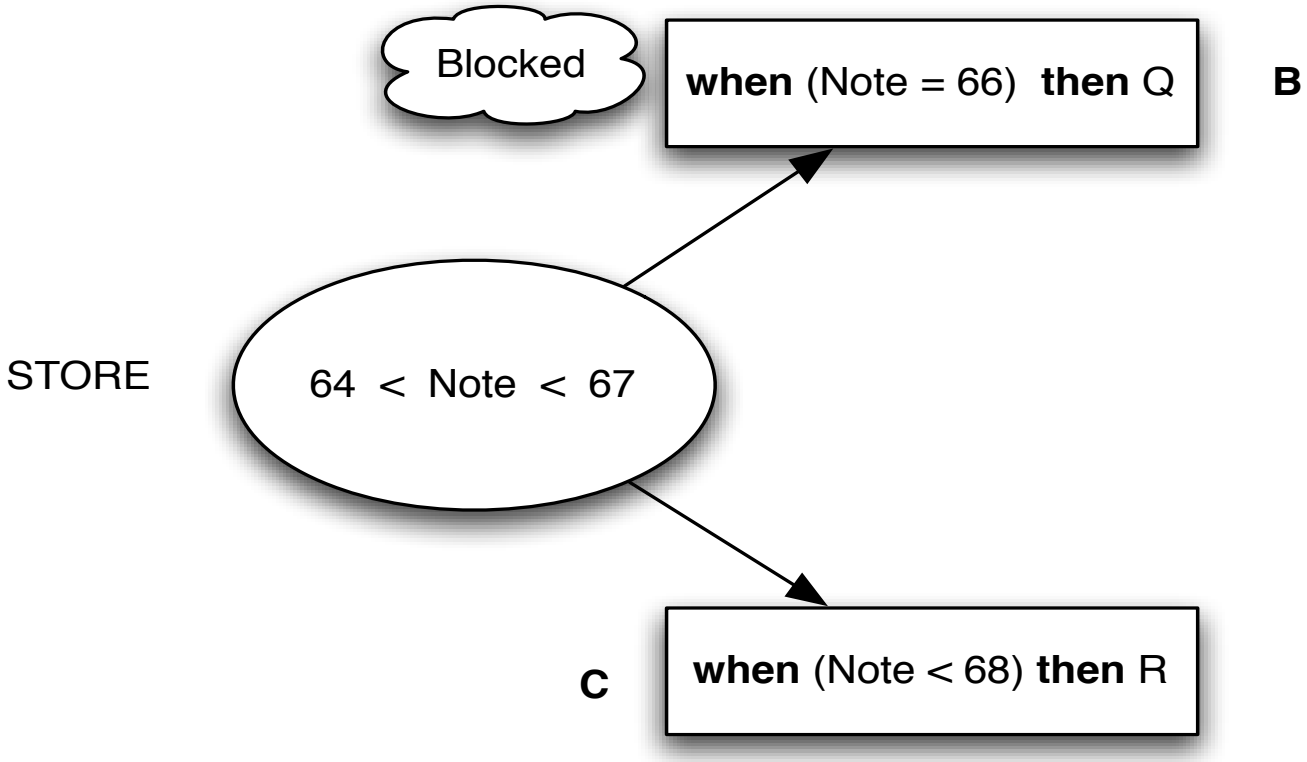
Computation in ccp

ccp's store-as-constraint vs Von Neumann's store-as-valuation



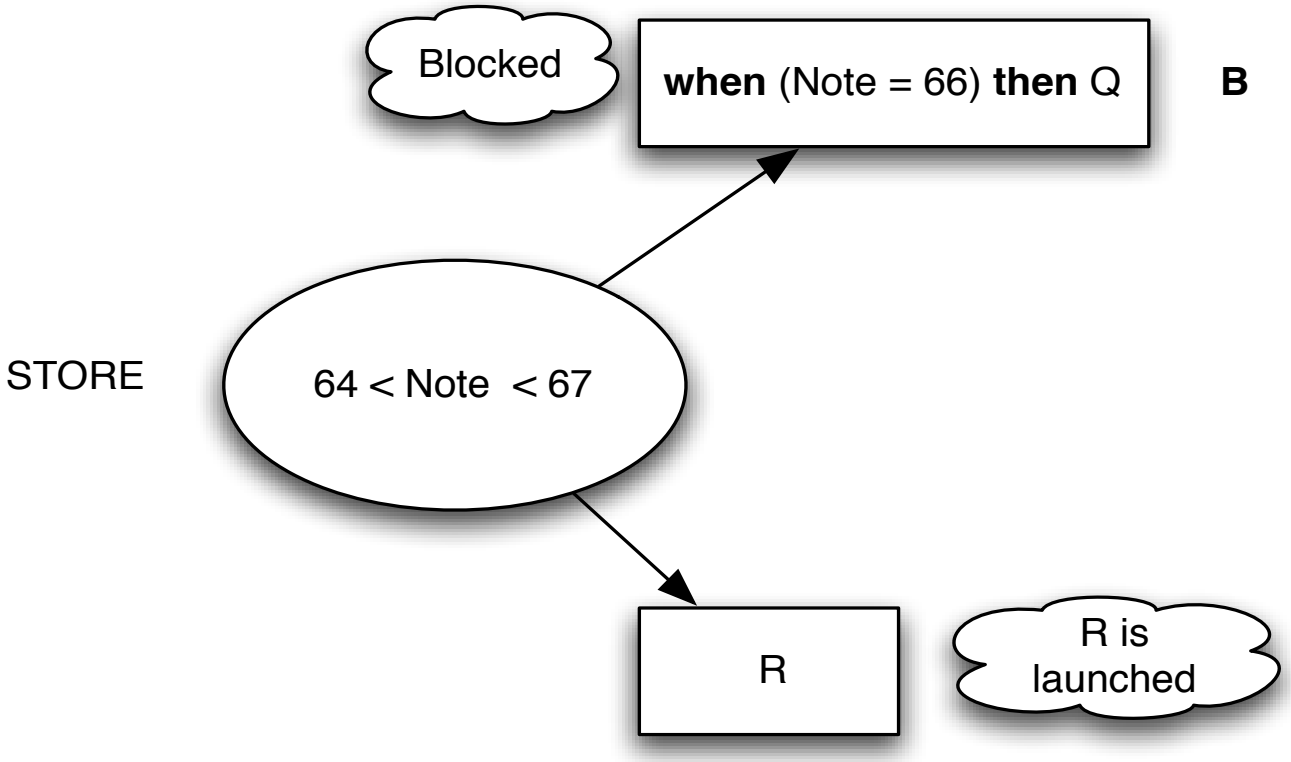
Computation in ccp

ccp's store-as-constraint vs Von Neumann's store-as-valuation

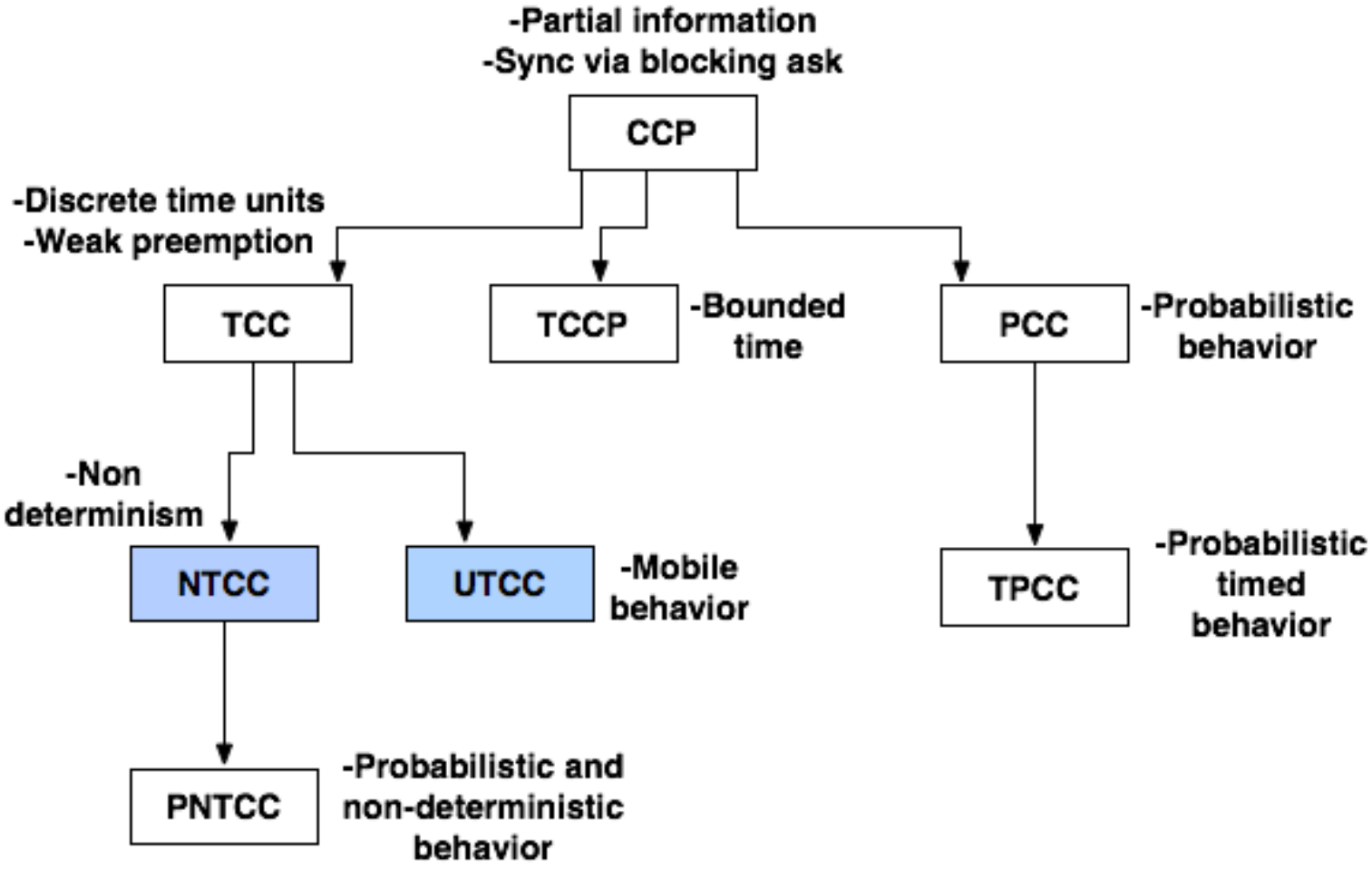


Computation in ccp

ccp's store-as-constraint vs Von Neumann's store-as-valuation



ccp a family of calculi



Outline

ntcc

For specifying timed reactive systems

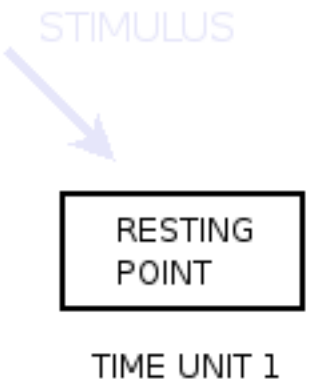
Concurrent processes communicating via asynchronous channels

- ccp + ideas from synchronous languages:
  - computation proceeds in discrete time units
- Considers negative information, “an event did not happen”
- and choice, “select one of a given set of actions”

The ntcc Model

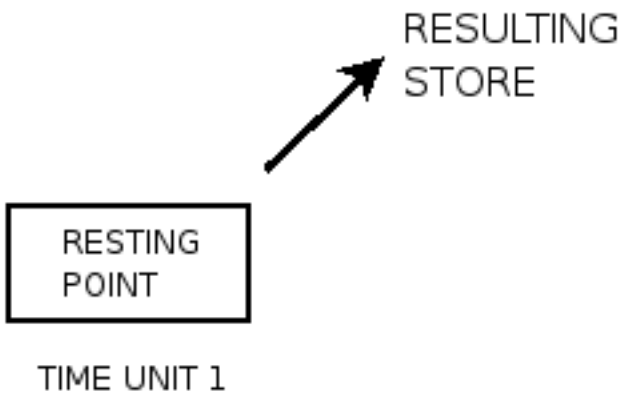
1. Receives a stimulus (i.e a constraint) from the environment.

The ntcc Model



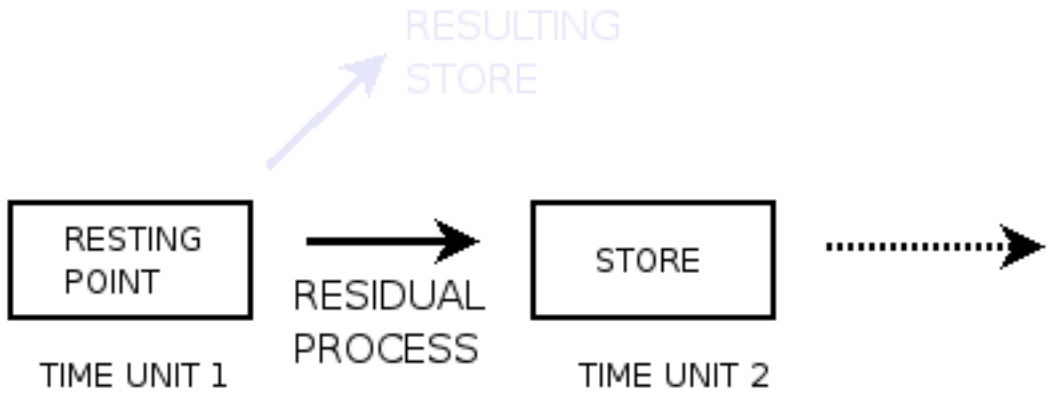
1. Receives a stimulus (i.e a constraint) from the environment.
2. Computes a CCP process in the current time-unit and wait for stability.

The ntcc Model



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.

The ntcc Model



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 ntcc calculus

Syntax

$$P, Q := \quad \mathbf{skip} \mid \mathbf{tell}(c) \mid P \parallel Q \mid \sum_{i \in S} \mathbf{when}\ c_i\ \mathbf{do}\ P_i \mid (\mathbf{local}\, \vec{x})\, P \mid$$
$$\mathbf{next}\, P \mid \mathbf{unless}\ c\ \mathbf{next}\, P \mid \star\, P \mid !\, P$$

- $\mathbf{tell}(c)$: adds constraint c to the store in the current time interval.
- $P \parallel Q$ is the concurrent execution of P and Q
- $\sum_{i \in S} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$: chooses some P_i such that c_i can be deduced from the current store.
- $(\mathbf{local}\, \vec{x}; c)\, P$: behaves like P but the information about variables in $\vec{x}$ is local to P

The ntcc calculus

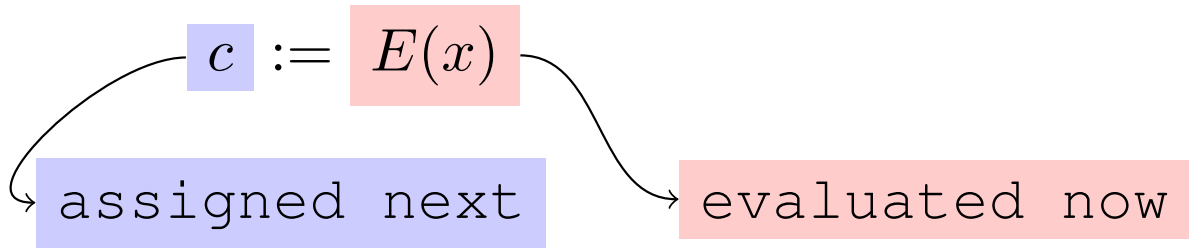
Syntax

$$P, Q := \quad \mathbf{skip} \mid \mathbf{tell}(c) \mid P \parallel Q \mid \sum_{i \in S} \mathbf{when}\ c_i\ \mathbf{do}\ P_i \mid (\mathbf{local}\, \vec{x})\, P \mid$$
$$\mathbf{next}\, P \mid \mathbf{unless}\ c\ \mathbf{next}\, P \mid \star\, P \mid !\, P$$

- $\mathbf{next}\, P$: executes P in the next time unit.
- $\mathbf{unless}\ c\ \mathbf{next}\, P$: executes P in the next time unit if c cannot be entailed now.
- $\star\, P$: executes P eventually.
- $!\, P$: executes P now and in the future.

Some derived constructs

- Cells: assignable variables with persistent values



- Procedure definition:
  $f(i) = P$
  restriction: recursive calls within a **next** construct

An example

- A system observing occurrence of events
- At some beat, if the waited for event occurs, k actions spaced m beats are launched,
- if the event does not occur, actions are launched at the next beat

Simple example: launching a group of actions

$$Set\_tempo = !\,(\mathbf{when}\ st = 0\ \mathbf{do}\ \mathbf{tell}(st := tempo) \parallel \mathbf{tell}(beat) \parallel \mathbf{tell}(bc := bc + 1) \parallel \mathbf{when}\ st > 0\ \mathbf{do}\ \mathbf{tell}(st := st - 1))$$

$$Group(i,k,m) = !\,\mathbf{when}\ beat \wedge bc = i\ \mathbf{do}\ \mathbf{when}\ event\ \mathbf{do}\ Launch(i,0,k,m) \parallel \mathbf{unless}\ event\ \mathbf{next}\ Launch(i+1,0,k,m)$$

$$Launch(i,c,k,m) = \mathbf{when}\ beat\ \mathbf{do}\ \mathbf{when}\ bc = i + c * m\ \mathbf{do}\ \mathbf{tell}(launched) \parallel \mathbf{unless}\ c = k\ \mathbf{next}\ Launch(i,c+1,k,m) \parallel \mathbf{unless}\ bc = i + c * m\ \mathbf{next}\ Launch(i,c,k,m) \parallel \mathbf{unless}\ beat\ \mathbf{next}\ Launch(i,c,k,m)$$

$$System = Set\_tempo \parallel \mathbf{tell}(st = 0) \parallel \mathbf{tell}(bc = 0) \parallel Group(3,2,2)$$

Formal semantics of ntcc

- Operational semantics: reduction rules over configurations.

$$\mathrm{R_T}\ \frac{}{\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{skip}, d \wedge c\rangle}$$

- Denotational semantics:
  - What is observed of a process: the sequence of its output stores (constraints) $\alpha = c_1 c_2 ...$
  - Semantics of P: all sequences it outputs for any input

$$sp(P) = \{\alpha' \mid P \xRightarrow{(\alpha,\alpha')}{}^{\omega} \ for\ some\ \alpha\}$$

Proving properties of processes

- View processes as formulae in linear temporal logic (LTL)

$$A, B, \ldots := c \mid A \Rightarrow A \mid \dot{\neg} A \mid \dot{\exists}_x A \mid \circ A \mid \Box A \mid \Diamond A$$

- Then, for a property F to be verified of a process P, prove $P \models_{LTL} F$.

Proving properties of processes

- View processes as formulae in linear temporal logic (LTL)

$$A, B, \ldots := c \mid A \Rightarrow A \mid \dot{\neg} A \mid \dot{\exists}_x A \mid \circ A \mid \Box A \mid \Diamond A$$

- Then, for a property F to be verified of a process P, prove $P \models_{LTL} F$.

There is a proof procedure to verify properties expressed as LTL formulae

... but only for “locally independent processes”

ntcc proof system (partial)

$$\mathbf{tell}(c) \vdash c$$

$$\frac{\forall i \in I \quad P_i \vdash A_i}{\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i \vdash \dot{\bigvee}_{i \in I} (c_i \dot{\wedge} A_i) \dot{\vee} \dot{\bigwedge}_{i \in I} \dot{\neg} c_i}$$

$$\frac{P \vdash A \qquad Q \vdash B}{P \parallel Q \vdash A \dot{\wedge} B}$$

$$\frac{P \vdash A}{(\mathbf{local}\, x)\, P \vdash \dot{\exists}_x A}$$

$$\frac{P \vdash A}{\mathbf{next}\, P \vdash \circ A}$$

Outline

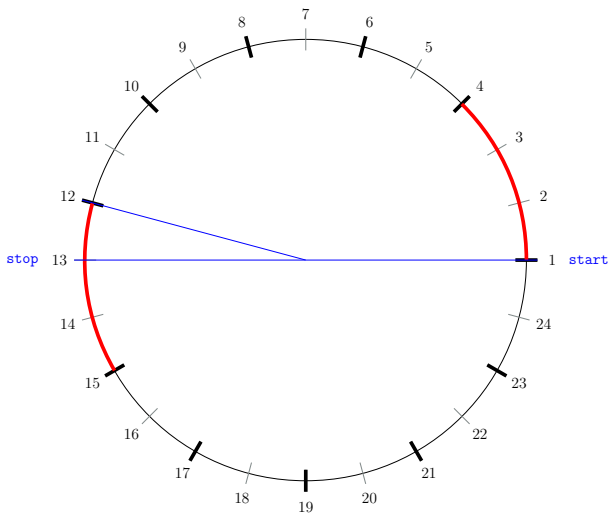
Music example: rhythm patterns

Rhythmic patterns of Central African Republic (M. Chemillier).

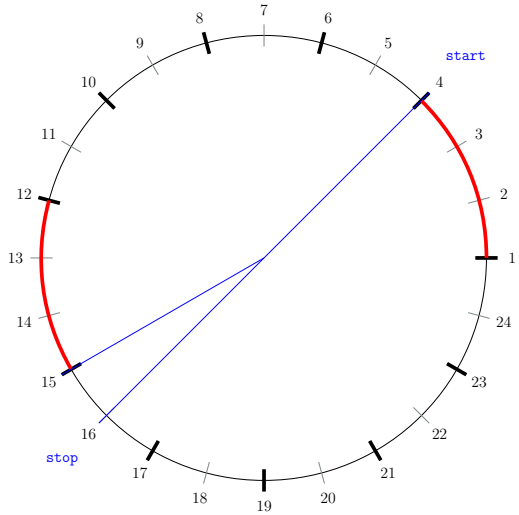
Patterns: two-beat groups separated by 3-beat elements:

$$3\ \ 2\,2\,2\,2\ \ 3\ \ 2\,2\,2\,2\,2$$

Playing can be started at any position in the sequence:



(a)



(b)

ntcc example: the model

$$Beatp \stackrel{\text{def}}{=} \mathbf{tell}(beat) \parallel \prod_{i \in I_i} \mathbf{next}^{\,i}\mathbf{tell}(beat)$$

$$Startp \stackrel{\text{def}}{=} \mathbf{tell}(start) + \sum_{i \in I_2} \mathbf{next}^{\,i}(\mathbf{tell}(start))$$

$$Check \stackrel{\text{def}}{=} !\,\mathbf{when}\ start\ \mathbf{do}\ \mathbf{next}^{\,12}(\mathbf{tell}(stop))$$

$$System \stackrel{\text{def}}{=} Beatp \parallel Startp \parallel Check$$

where $I_1 = \{3, 5, 7, 9, 11, 14, 16, 18, 20, 22\}$
and $I_2 = \{3, 5, 7, 9, 11\}$

Asymmetry property:

"cannot break the circle of the pattern in two equal parts"

ntcc example: proofs

Encoding

$$
\begin{aligned}
[\![Beatp]\!] &= beat \,\dot{\wedge} \dot{\bigwedge_{i \in I_1}} \circ^i beat \\
[\![Startp]\!] &= start \,\dot{\vee} \dot{\bigvee_{i \in I_2}} \circ^i start \\
[\![Check]\!] &= \Box(start \,\dot{\Rightarrow} \circ^{12} stop) \\
[\![System]\!] &= [\![Beat]\!] \,\dot{\wedge} [\![Start]\!] \,\dot{\wedge} [\![Check]\!]
\end{aligned}
$$

Asymmetry property:

$$[\![System]\!] \models \Diamond(start \,\dot{\wedge} \circ^{11}(beat \,\dot{\wedge} \circ stop))$$

ntcc example: a refinement

Explore the relation between the placement of the 3 in the beat pattern and the asymmetry property.

$$Beat' \stackrel{\texttt{def}}{=} \mathbf{tell}(beat) \parallel \mathbf{next}^{\,3} \sum_{i \in I_3} (\mathbf{tell}(pos = i) \parallel Beat\_Aux(i-1))$$

$$\begin{aligned} Beat\_Aux(N) \stackrel{\texttt{def}}{=} \; & \mathbf{tell}(beat) \parallel \\ & \mathbf{when}\ N = 1\ \mathbf{do}\ \mathbf{next}^{\,3} Beat\_Aux(0) \\ + & \mathbf{when}\ N \neq 1\ \mathbf{do}\ \mathbf{next}^{\,2} Beat\_Aux(N-1) \end{aligned}$$

$$System' \stackrel{\texttt{def}}{=} Beat' \parallel Start \parallel Check$$

where $I_3 = \{2, 3, 4, 5, 6\}$

ntcc example: properties

Explore the relation between the placement of the 3 in the beat pattern and the asymmetry property ($I_3 = \{2, 3, 4, 5, 6\}$).

$$
\begin{aligned}
Beat' &\stackrel{\texttt{def}}{=} \mathbf{tell}(beat) \parallel \mathbf{next}^{\,3} \sum_{i \in I_3} (\mathbf{tell}(pos = i) \parallel Beat\_Aux(i-1)) \\
Beat\_Aux(N) &\stackrel{\texttt{def}}{=} \mathbf{tell}(beat) \parallel \\
&\qquad \mathbf{when}\ N = 1\ \mathbf{do}\ \mathbf{next}^{\,3} Beat\_Aux(0) \\
&\qquad + \mathbf{when}\ N \neq 1\ \mathbf{do}\ \mathbf{next}^{\,2} Beat\_Aux(N-1) \\
System' &\stackrel{\texttt{def}}{=} Beat' \parallel Start \parallel Check
\end{aligned}
$$

$$[\![System']\!] \models \Diamond((pos = x) \dot{\Rightarrow} \Diamond(stop \,\dot{\wedge}\, \circ beat))$$

Outline

Dynamic interactive structures

Movable hierarchical structures containing interaction points

"mobility" is understood as in the π-calculus: communication of links (private variables) between processes.

`This cannot be expressed in ntcc`

A ntcc with mobility: utcc

The `utcc` calculus (C. Olarte) replaces ntcc construct $\mathbf{when}\ c\ \mathbf{do}\ P$ by $(\mathbf{abs}\ \vec{x}; c)\ P$

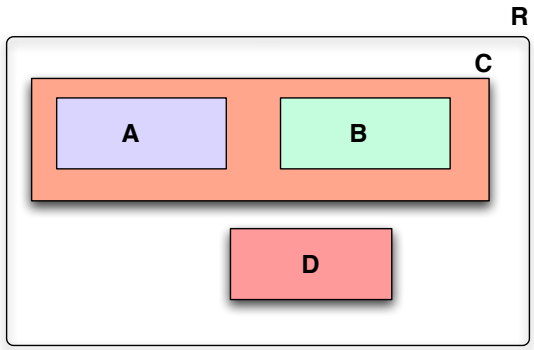
$(\mathbf{abs}\ \vec{x}; c)\ P$: executes $P[\vec{t}/\vec{x}]$ for each $\vec{t}$ s.t. $c[\vec{t}/\vec{x}]$ can be deduced from the current store.

Communicating private link a thru channel ch:

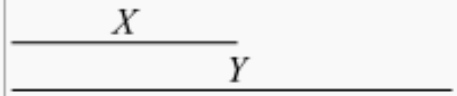
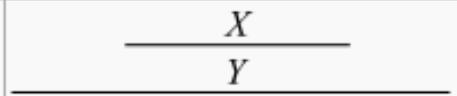
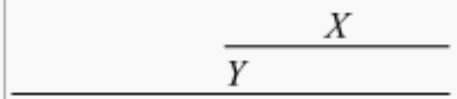
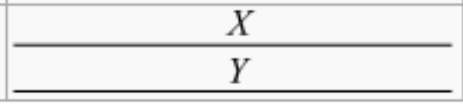
$$(\mathbf{local}\ a)\ \mathbf{tell}(ch(a)) \parallel (\mathbf{abs}\ \vec{x}; ch(x))\ P$$

Interactive Scores

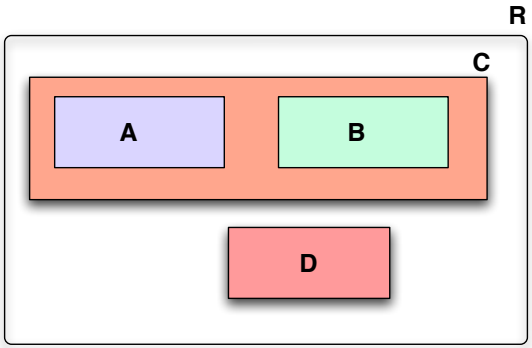
(M. Desainte-Catherine)



Allen relations:

Relation	Illustration	Interpretation
$X<Y$ $Y>X$		X takes place before Y
$X\mathrm{m}Y$ $Y\mathrm{mi}X$		X meets Y (*i* stands for *inverse*)
$X\mathrm{o}Y$ $Y\mathrm{oi}X$		X overlaps with Y
$X\mathrm{s}Y$ $Y\mathrm{si}X$		X starts Y
$X\mathrm{d}Y$ $Y\mathrm{di}X$		X during Y
$X\mathrm{f}Y$ $Y\mathrm{fi}X$		X finishes Y
$X=Y$		X is equal to Y

Interactive Scores

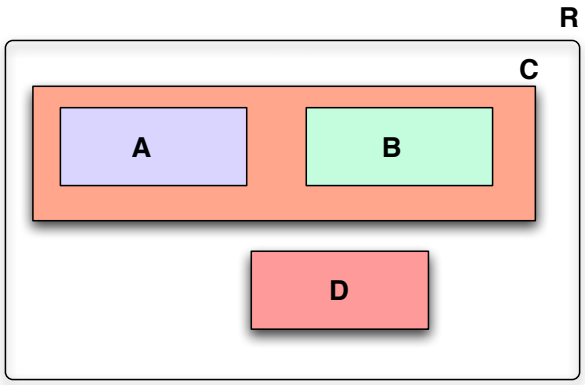


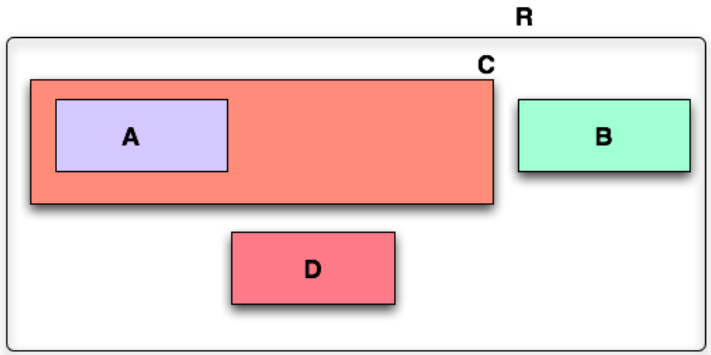
Score relations:

A precedes B	A meets D
D overlaps B	R contains C
R contains D	C contains A

- An Interactive Score is a pair composed of temporal objects and structural and temporal (Allen) relations.
- Each object is comprised of a start-time, a duration, and a procedure (operational meaning).
- The idea: Dynamic changes in the hierarchy. E.g., if an interaction does not occur, the composer may move the interval to a similar musical context.

Dynamic Interactive Scores





Dynamic Reconfiguration:

- Moving boxes.
- Adding/deleting intervals.

The Model

$$
\begin{aligned}
BoxOp \stackrel{\texttt{def}}{=}\ & (\textbf{abs}\ id, d; \texttt{mkbox}(id, d))\ (\textbf{local}\ s)\ \textbf{tell}(\texttt{box}(id, d, s)) \\
\| \ & (\textbf{abs}\ id; \texttt{destroy}(id)) \\
& \quad (\textbf{abs}\ x, sup; \texttt{in}(x, id) \wedge \texttt{in}(id, sup)) \\
& \quad\ \textbf{unless}\ \texttt{play}(id)\ \textbf{next}\ \textbf{tell}(\texttt{in}(x, sup)) \\
\| \ & (\textbf{abs}\ x, y; \texttt{before}(x, y))\ \textbf{when}\ \exists_z(\texttt{in}(x, z) \wedge \texttt{in}(y, z))\ \textbf{do} \\
& \quad \textbf{unless}\ \texttt{play}\ (y)\ \textbf{next}\ \textbf{tell}(\texttt{bf}(x, y)) \\
\| \ & (\textbf{abs}\ x, y; \texttt{into}(x, y))\ \textbf{unless}\ \texttt{play}\ (x)\ \textbf{next}\ \textbf{tell}(\texttt{in}(x, y)) \\
\| \ & (\textbf{abs}\ x, y; \texttt{out}(x, y))\ \textbf{when}\ \texttt{in}(x, y)\ \textbf{do} \\
& \quad \textbf{unless}\ \texttt{play}\ (x)\ \textbf{next}\ (\textbf{abs}\ z, \texttt{in}(y, z); \textbf{tell}(\texttt{in}(x, z)))
\end{aligned}
$$

The Model

$$
\begin{aligned}
BoxOp \stackrel{\text{def}}{=}\ & (\mathbf{abs}\ id, d; \texttt{mkbox}(id, d))\ (\mathbf{local}\ s)\ \mathbf{tell}(\texttt{box}(id, d, s)) \\
& \parallel (\mathbf{abs}\ id; \texttt{destroy}(id)) \\
& \qquad (\mathbf{abs}\ x, sup; \texttt{in}(x, id) \wedge \texttt{in}(id, sup)) \\
& \qquad\quad \mathbf{unless}\ \texttt{play}(id)\ \mathbf{next}\ \mathbf{tell}(\texttt{in}(x, sup)) \\
& \parallel (\mathbf{abs}\ x, y; \texttt{before}(x, y))\ \mathbf{when}\ \exists_z(\texttt{in}(x, z) \wedge \texttt{in}(y, z))\ \mathbf{do} \\
& \qquad \mathbf{unless}\ \texttt{play}\ (y)\ \mathbf{next}\ \mathbf{tell}(\texttt{bf}(x, y)) \\
& \parallel (\mathbf{abs}\ x, y; \texttt{into}(x, y))\ \mathbf{unless}\ \texttt{play}\ (x)\ \mathbf{next}\ \mathbf{tell}(\texttt{in}(x, y)) \\
& \parallel (\mathbf{abs}\ x, y; \texttt{out}(x, y))\ \mathbf{when}\ \texttt{in}(x, y)\ \mathbf{do} \\
& \qquad \mathbf{unless}\ \texttt{play}\ (x)\ \mathbf{next}\ (\mathbf{abs}\ z, \texttt{in}(y, z); \mathbf{tell}(\texttt{in}(x, z)))
\end{aligned}
$$

$$
\begin{aligned}
Const \stackrel{\text{def}}{=}\ & (\mathbf{abs}\ x, y; \texttt{in}(x, y))\ (\mathbf{abs}\ d_x, s_x; \texttt{box}(x, d_x, s_x)) \\
& \qquad (\mathbf{abs}\ d_y, s_y; \texttt{box}(y, d_y, s_y)) \\
& \qquad\quad \mathbf{tell}(s_y \leq s_x) \parallel \mathbf{tell}(d_x + s_x \leq d_y + s_y) \\
& \parallel (\mathbf{abs}\ x, y; \texttt{bf}(x, y))\ (\mathbf{abs}\ d_x, s_x; \texttt{box}(x, d_x, s_x)) \\
& \qquad (\mathbf{abs}\ d_y, s_y; \texttt{box}(y, d_y, s_y))\ \mathbf{tell}(s_x + d_x \leq s_y)
\end{aligned}
$$

The Model(2)

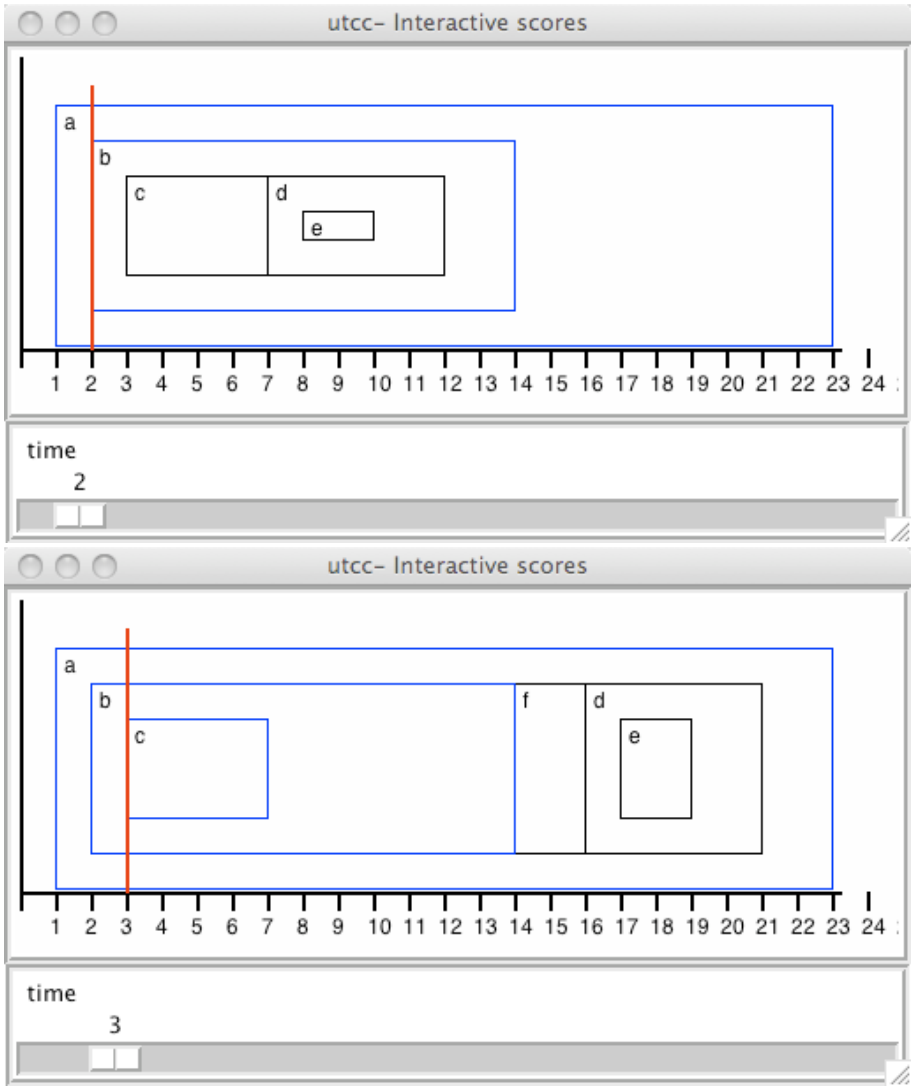
$$Clock(t,v) \stackrel{\text{def}}{=} \mathbf{tell}(t = v) \parallel \mathbf{next}\, Clock(t, v+1)$$

$$Play(x,t) \stackrel{\text{def}}{=} \mathbf{when}\ t \geq 1\ \mathbf{do}\ \mathbf{tell}(\mathtt{play}(x)) \parallel \mathbf{unless}\ t \leq 1\ \mathbf{next}\, Play(x, t-1)$$

$$Init(t) \stackrel{\text{def}}{=} (\mathbf{wait}\ x; init(x))\ \mathbf{do}\ (\mathbf{abs}\ d_x, s_x; \mathtt{box}(x, d_x, s_x)) \; Clock(t,0) \parallel \mathbf{tell}(s_x = t) \parallel !\,(\mathbf{wait}\ y, d_y, s_y; \mathtt{box}(y, d_y, s_y) \wedge s_y \leq t)\ \mathbf{do}\ Play(y, d_y)$$

$$System \stackrel{\text{def}}{=} (\mathbf{local}\, t)\, Init(t) \parallel\!! Constraints \parallel\!! BoxOp \parallel UsrBoxes$$

An Example



$$UsrBoxes \stackrel{\text{def}}{=}$$

$$\begin{aligned}
&\textbf{tell}(\texttt{mkbox}(a, 22)) \parallel \\
&\textbf{tell}(\texttt{mkbox}(b, 12)) \parallel \\
&\textbf{tell}(\texttt{mkbox}(c, 4)) \parallel \\
&\textbf{tell}(\texttt{mkbox}(d, 5)) \\
&\textbf{tell}(\texttt{mkbox}(e, 2)) \parallel \\
&\textbf{tell}(\texttt{into}(b, a)) \parallel \\
&\textbf{tell}(\texttt{into}(c, b)) \parallel \\
&\textbf{tell}(\texttt{into}(d, b)) \parallel \\
&\textbf{tell}(\texttt{into}(e, d)) \parallel \\
&\textbf{tell}(\texttt{before}(c, d)) \parallel \\
&\textbf{whenever } \texttt{play}(b) \textbf{ do} \\
&\quad \textbf{unless } \texttt{signal} \textbf{ next} \\
&\qquad \textbf{tell}(\texttt{out}(d, b)) \parallel \\
&\qquad \textbf{tell}(\texttt{mkbox}(f, 2)) \parallel \\
&\qquad \textbf{tell}(\texttt{into}(f, a)) \parallel \\
&\qquad \textbf{tell}(\texttt{before}(b, f) \parallel \\
&\qquad \textbf{tell}(\texttt{before}(f, d)))
\end{aligned}$$

Declarative Interpretation of utcc

- Processes defined by the user may lead to inconsistent stores: E.g. placing a box that exceeds the boundaries of the container.
- The idea: Using the declarative view of utcc processes as FLTL formulae to verify the model.

Definition (utcc logic characterization)

$$
\begin{array}{lcllcl}
[\![\mathbf{skip}]\!] & = & \texttt{true} & [\![\mathbf{tell}(c)]\!] & = & c \\
[\![P \parallel Q]\!] & = & [\![P]\!] \wedge [\![Q]\!] & [\![(\mathbf{abs}\ \vec{y}; c)\ P]\!] & = & \forall \vec{y}(c \Rightarrow [\![P]\!]) \\
[\![(\mathbf{local}\ \vec{x}; c)\ P]\!] & = & \exists \vec{x}(c \wedge [\![P]\!]) & [\![\mathbf{next}\ P]\!] & = & \circ [\![P]\!] \\
[\![\mathbf{unless}\ c\ \mathbf{next}\ P]\!] & = & c \vee \circ [\![P]\!] & [\![!\ P]\!] & = & \Box [\![P]\!]
\end{array}
$$

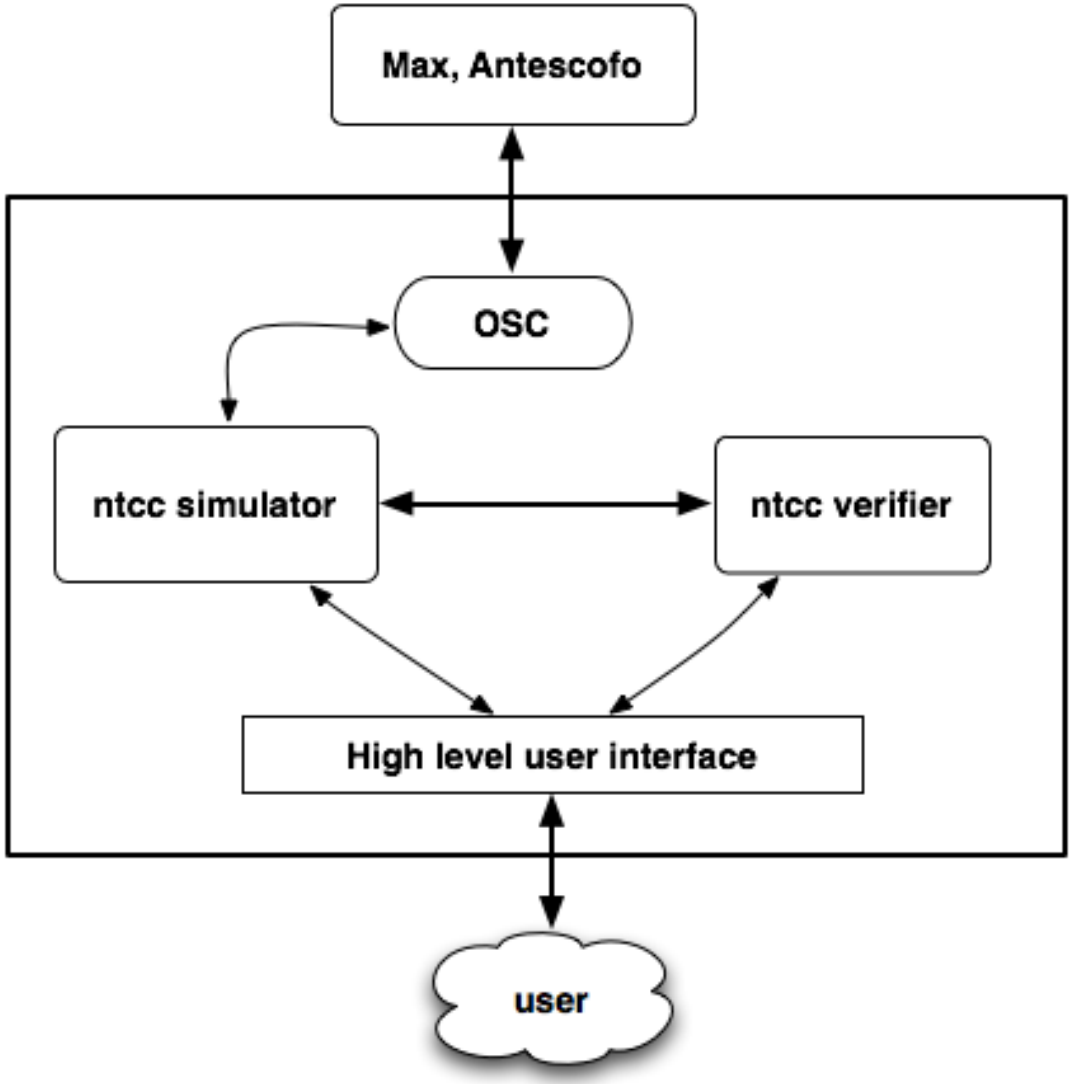
Verification of the Model

We can verify, for example,

- $[\![P]\!] \models \Diamond \exists_{x,d_x,s_x,y,d_y,s_y} (\texttt{box}(x, d_x, s_x) \wedge \texttt{box}(y, d_y, s_y) \wedge \texttt{in}(x, y) \wedge s_x + d_x > s_y + d_y)$: The end time of the box y is less than the end time of the inner box x. I.e., the box y cannot contain x.
- $[\![P]\!] \models \forall_x (\exists_{d_x,s_x} (\texttt{box}(x, d_x, s_x) \Rightarrow \Diamond \texttt{play}(x))$: All the musical structures are eventually played.
- $[\![P]\!] \models \Diamond \forall_x (\exists_{d_x,s_x} (\texttt{box}(x, d_x, s_x) \Rightarrow \texttt{play}(x))$: At some point all the boxes are playing simultaneously.

Outline

Tools: as in the “concurrency workbench”



Simulators

- sntcc, written in Mozart-Oz
  - Constraints: finite domains, reals (interval arithmetic)
  - $\mathbf{ask}, \mathbf{tell}$ constructs: derived directly from Oz instructions
  - Concurrency: Oz threads
  - Interface with music tools: none. In progress: OSC
- ntccrt (M. Toro), written in C++, with Gecode
  - Constraints: finite domains, finite sets
  - $\mathbf{tell}$: directly in Gecode. $\mathbf{ask}$: reified constraints.
  - Concurrency: threads as Gecode propagators
  - Interface with music tools: Max/MSP

($sntcc$ has been used in an application with $1,000,000$ time units)

sntcc simulator

$$player(i) = \textbf{when}\ beat\ \textbf{do}\ (\textbf{tell}(note(N_i)) + \textbf{skip}) \parallel \textbf{next}\ player(i+1) \parallel \textbf{unless}\ beat\ \textbf{next}\ player(i)$$

```
Player = fun lazy {$ I}
   par(when( proc{$ V} V.current.beat =: 1 end
             par(sum(tell( proc {$ V} V.current.note =: N.I end)
                     tell( proc {$ V} 1 =: 1 end))
                 next({Player I+1})))
   unless(proc{$ V} V.current.beat =: 1 end  {Player I}))
    end

Vars = var(beat: {FD.int 0#1}    note: FD.decl)
Res = {SNTCC.simulate [  {Player 0} ] Vars 100}
```

Model checkers

Strategies

- Translation of ntcc processes and LTL formula to Buchi automata
- Use appropriate bisimulation relation

Model checkers: Buchi automaton

Since each (restricted) ntcc process is equivalent to a Buchi automaton, to prove $P \models F$:

1. Encode LTL formula F as a ntcc process R_F
2. translate P and $(R_F \parallel P)$ to Buchi automata, $B(P), B(R_F)$
3. check language equivalence of both automata

Problems:
works for ntcc “locally independent processes”
Only a restricted form of negation is admitted for F
(current) complexity of translation algorithm is hyper-exponential

Model checkers: Bisimulation

1. define a suitable bisimulation relation for ntcc (done)
2. define the property as a ntcc process (done),
3. use an algorithm to verify bisimilarity

Outline

Future work

- Model in ntcc some synchronization strategies for Antescofo. Identify desirable properties.
- Integrate interfaces (OSC) to music applications for the Oz ntcc simulator
- Develop efficient Buchi translations for “bounded” versions of ntcc constructs
- Devise an algorithm for the ntcc process bisimilarity
- Develop a user “programming language” for the ntcc simulator+verifier

Thanks!