Formal Models of Timed Musical Processes
Doctoral Defense

Gerardo M. Sarria M.

Advisor: Camilo Rueda
Co-Advisor: Juan Francisco Diaz

Universidad del Valle
AVISPA Research Group

September 22, 2008

Motivation

Motivation

Music composition, performance and improvisation are complex tasks of defining and controlling real-time concurrent activities.

It is necessary a formal model capable of specifying real-time, concurrent, (a)sychronous and constrained systems to be used as a mathematical platform for musical composition and improvisation.

Motivation

Various formalisms have found use in practical musical situations.

Those formalisms were not intended originally to be used in music.

Process Calculi are popular contemporary formalisms for modeling and analyzing concurrent systems.

- Effective models of concurrency
- High degree of abstraction
- Intuitive “programming language” feel

Contributions

A new CCP calculus, called `rtcc`, for modeling real-time reactive systems.

- Strict extension of `ntcc`
- Explicit notions of time and resources
- Precise way of delay processes
- Strong preemption and default behaviour
- New approach to denotacional semantics (Chu Spaces)
- True Concurrency
- Intended for real-time multimedia interaction

Agenda

1. Preliminars
   - CCP
   - Constraint System
   - NTCC
2. The `rtcc` Calculus
   - Syntax
   - Operational Semantics
   - Denotational Semantics
   - Real-time Logics
3. Concluding Remarks and Future Work

Concurrent Constraint Programming

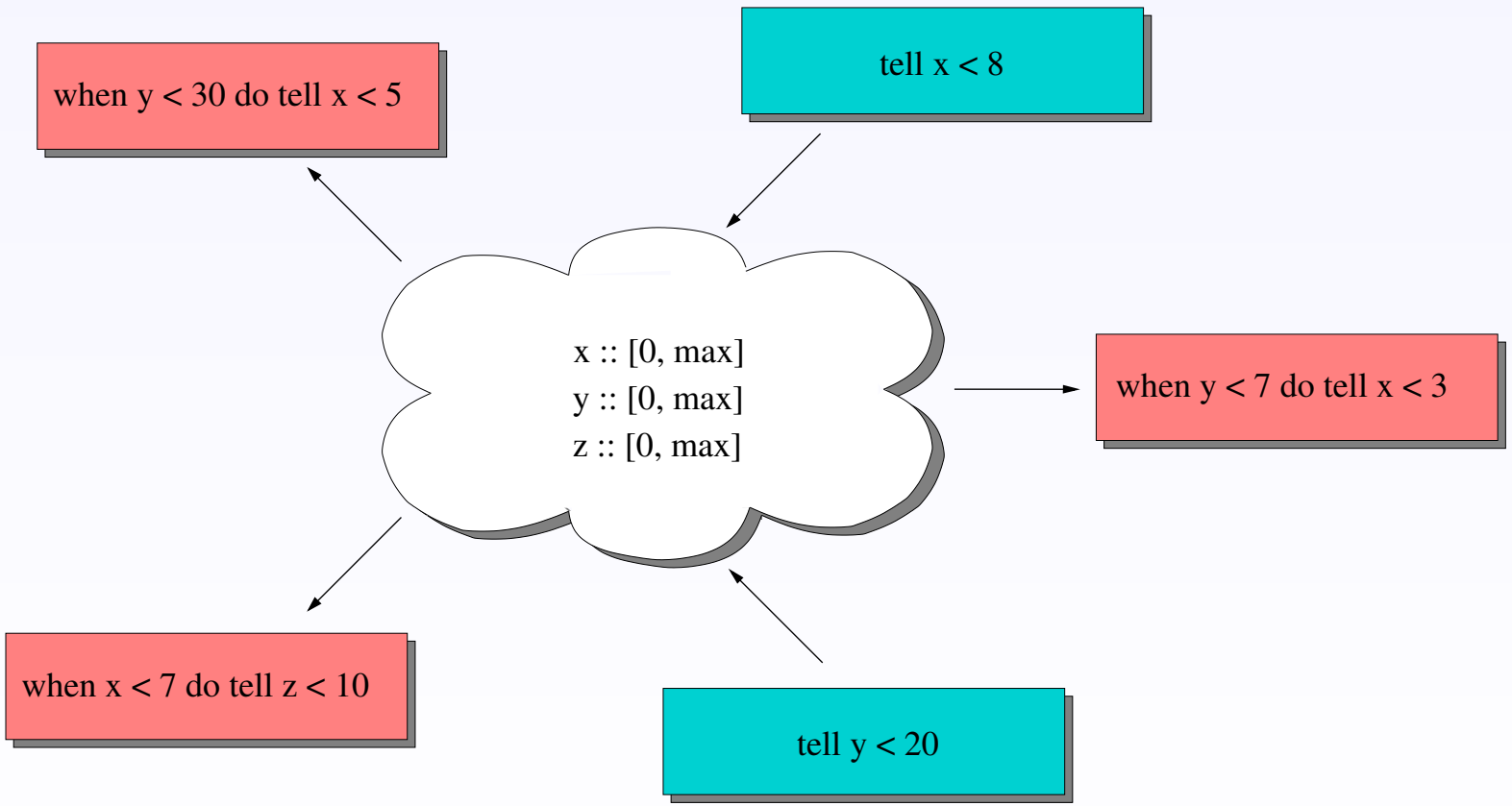
Concurrent constraint programming (CCP) is a model for specifying concurrent systems in terms of constraints.

A *constraint* is a first-order formulae representing partial information about shared variables.

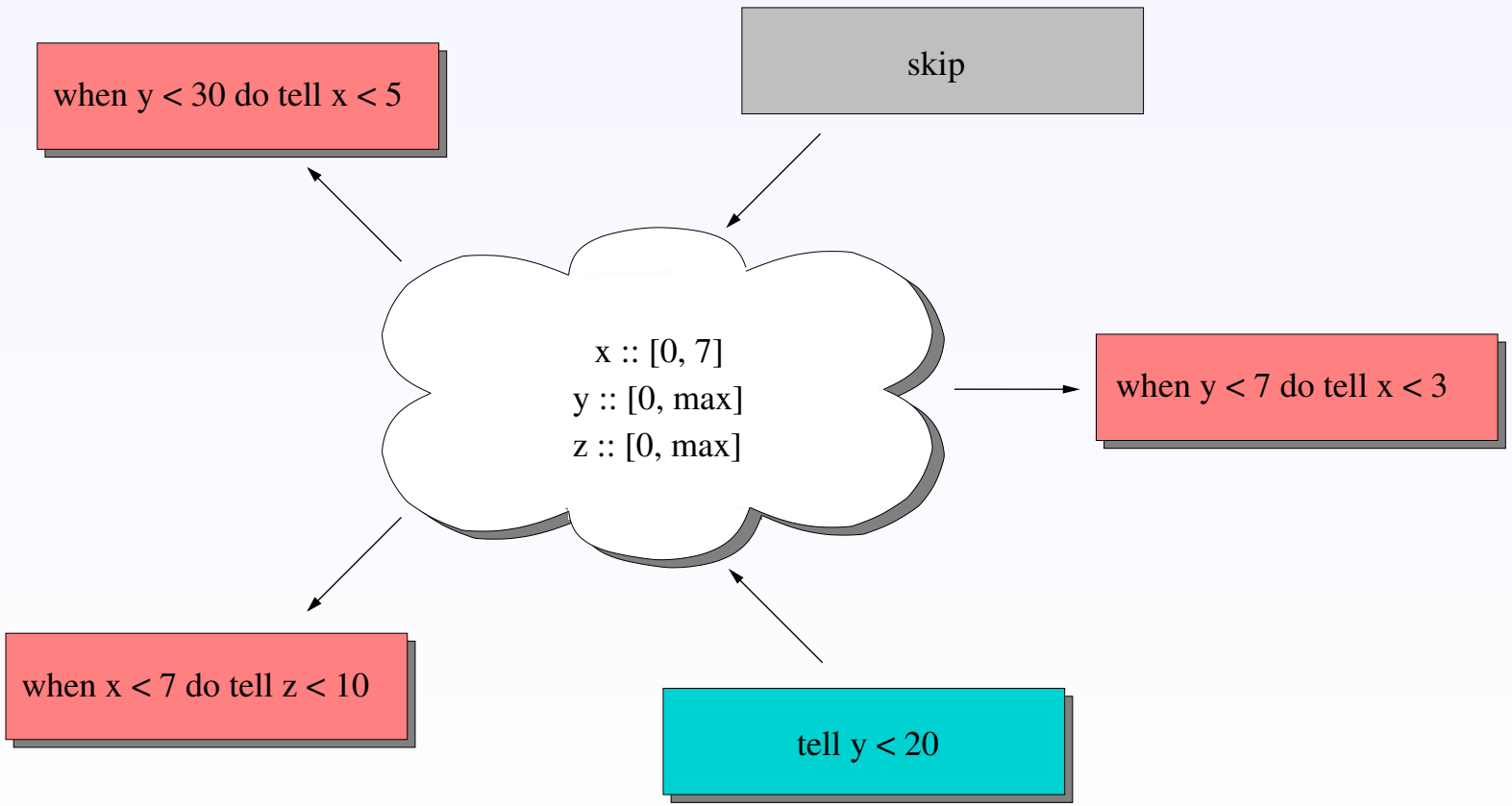
Von Neumann's store is replaced by a store of partial information (i.e. $x \geq 20$)

Read and write operators are replaced by `ask` and `tell`

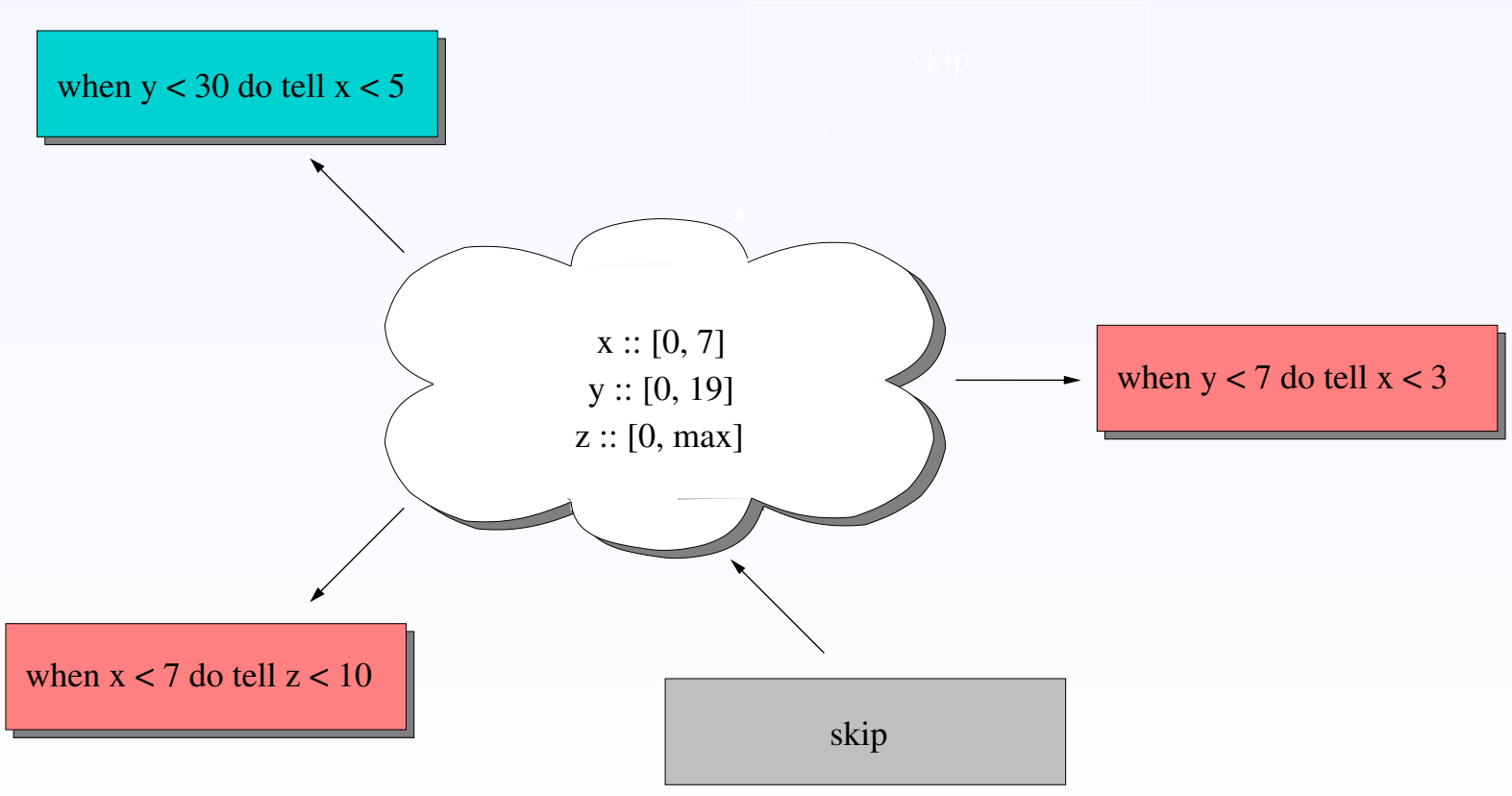
Concurrent Constraint Programming



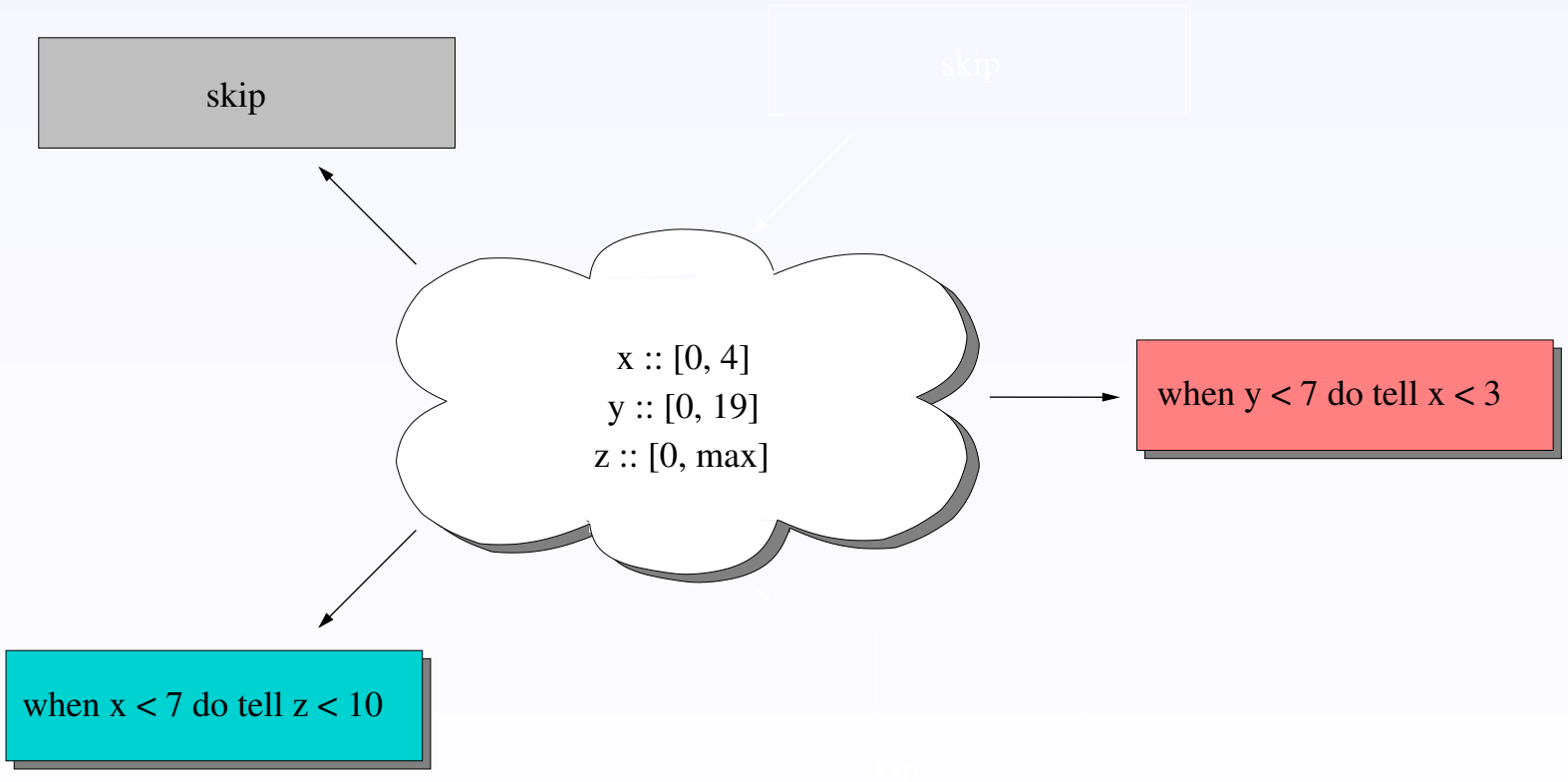
Concurrent Constraint Programming



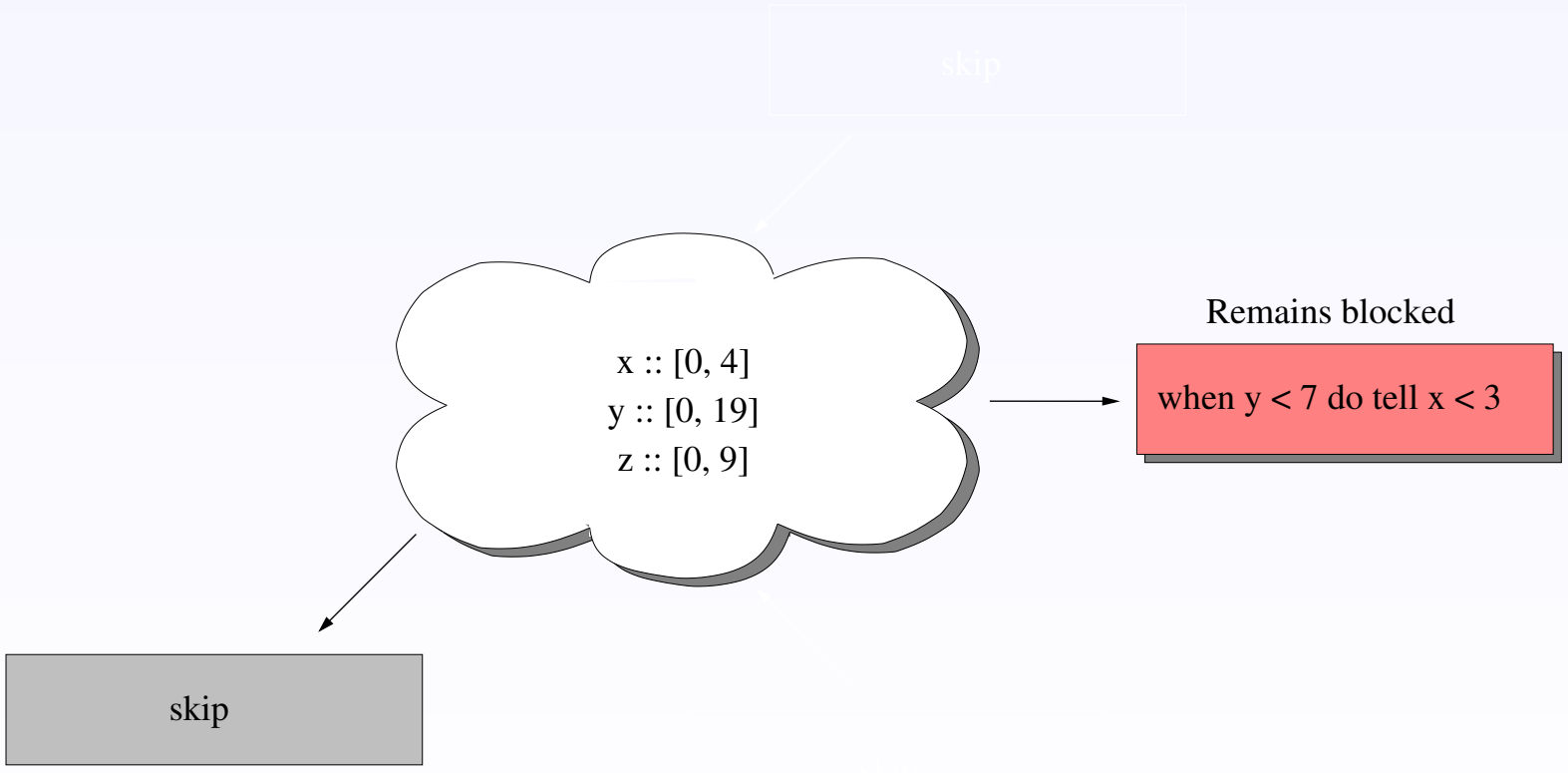
Concurrent Constraint Programming



Concurrent Constraint Programming



Concurrent Constraint Programming



Concurrent Constraint Programming

- Partial Information: Variables, Domains and Constraints
- Concurrency: Multiple agents
- Synchronization: via store

Constraint System

A constraint system specifies what kind of constraints handle the store.

Formally, it is a tuple $\langle \Sigma, \Delta \rangle$, where Σ is a signature (set of constraints, functions and predicate symbols) and Δ is a consistent first-order theory over Σ.

Constraints are first-order formulae over Σ

The conjunction of all constraints accumulated monotonically is called the store

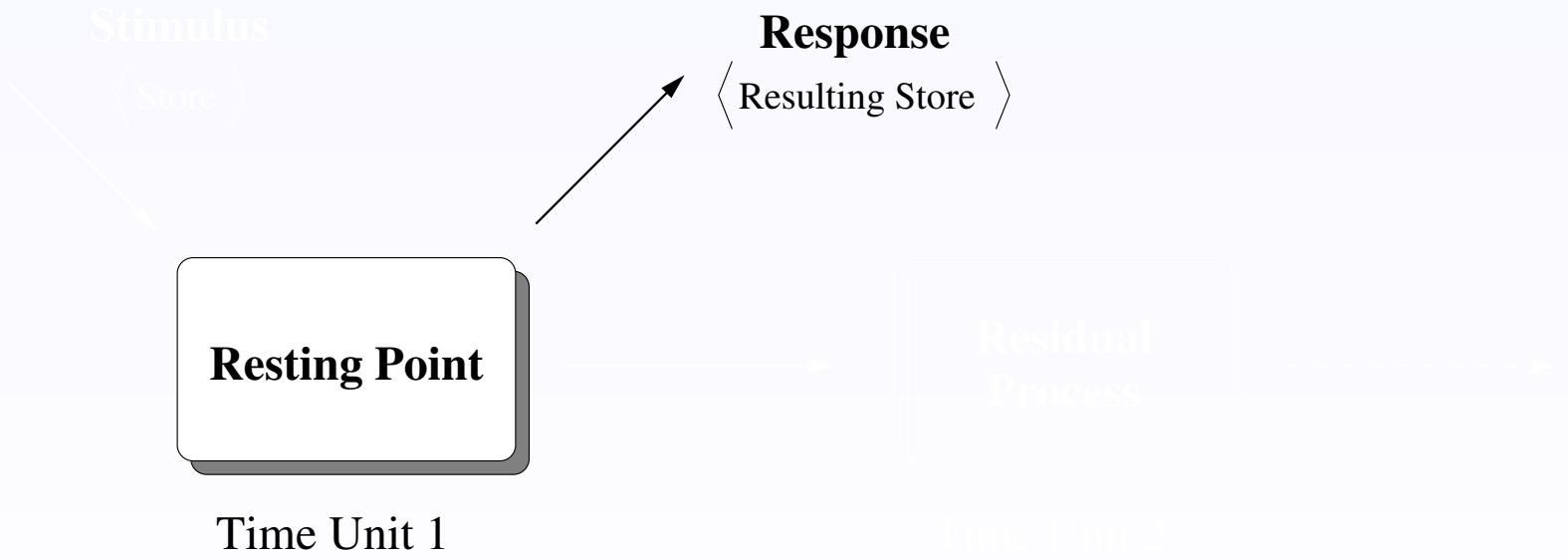
The entailment relation $c \vDash d$ holds iff $c \Rightarrow d$ is valid on Δ. $c \equiv d$ iff $c \vDash d$ and $d \vDash c$

The ntcc calculus is CCP calculi proposed for modeling and programming temporal reactive systems.

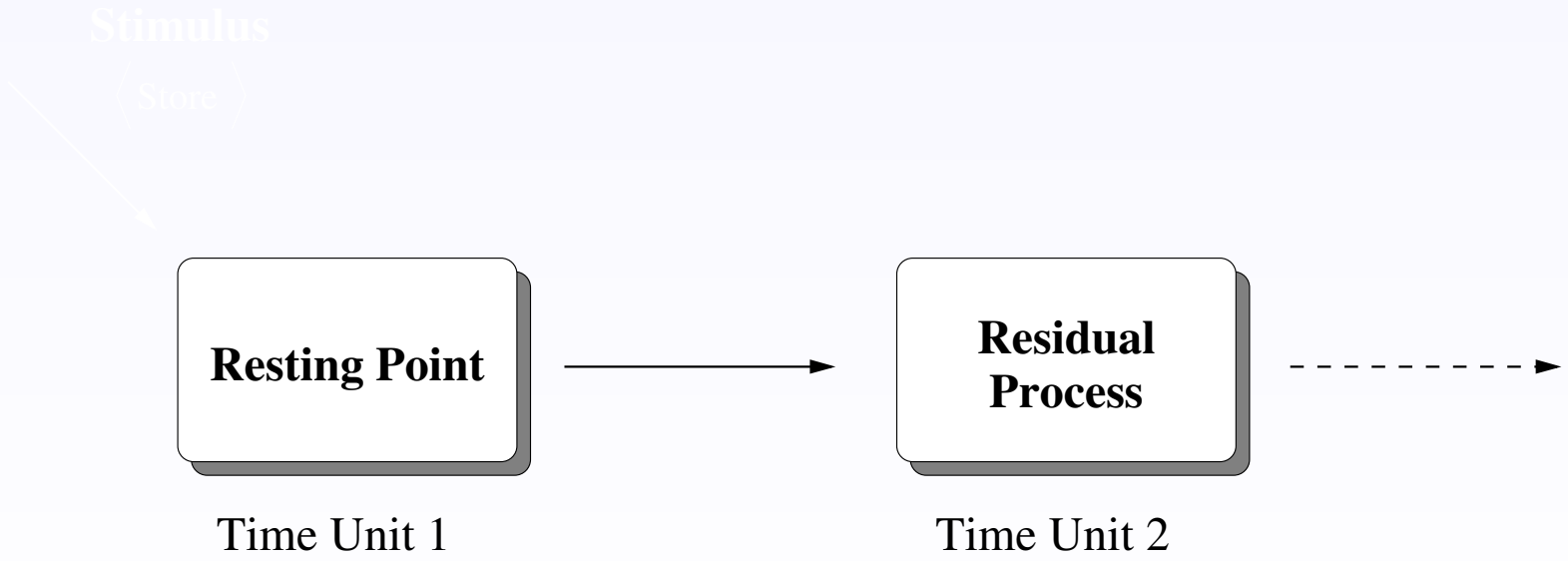
The ntcc calculus is CCP calculi proposed for modeling and programming temporal reactive systems.

The ntcc calculus is CCP calculi proposed for modeling and programming temporal reactive systems.

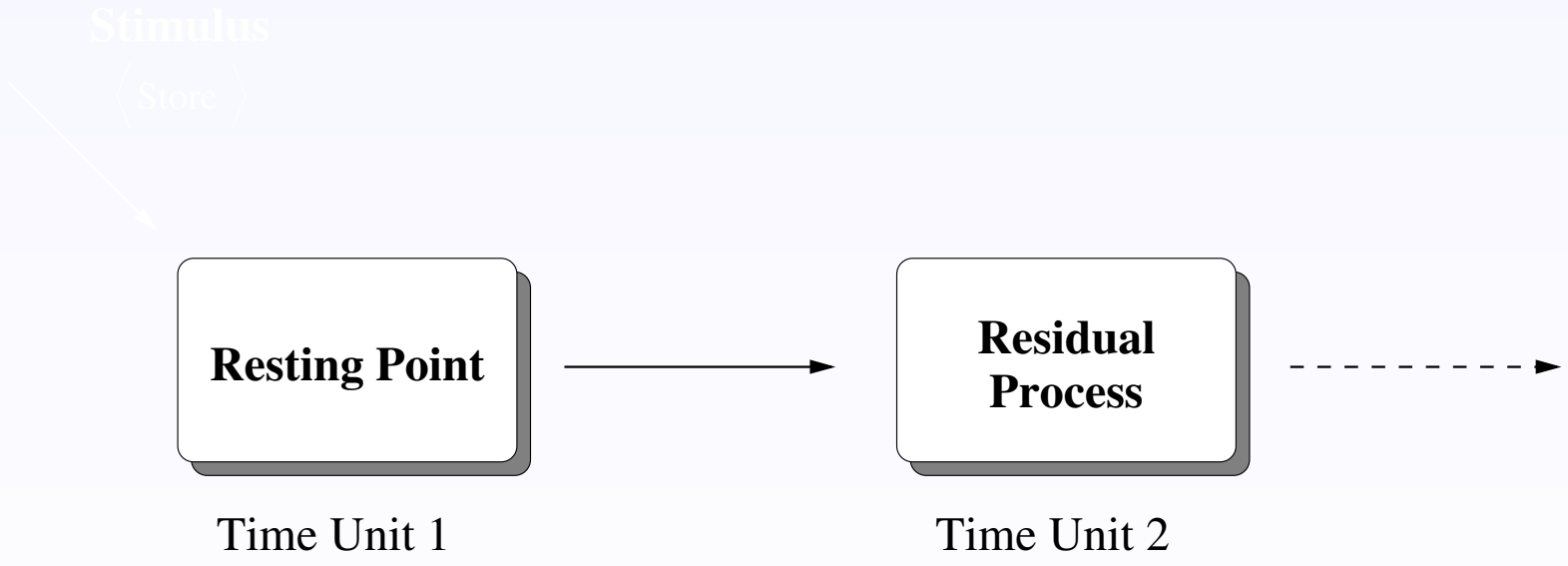
The ntcc calculus is CCP calculi proposed for modeling and programming temporal reactive systems.



The ntcc calculus is CCP calculi proposed for modeling and programming temporal reactive systems.



The ntcc calculus is CCP calculi proposed for modeling and programming temporal reactive systems.



Stores are not automatically transferred from a time unit to the next one.

Syntax

P, Q	::=	$\mathbf{tell}(c)$	Tell
	$\mid$	$\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$	Nondeterminism
	$\mid$	$P \parallel Q$	Parallel Composition
	$\mid$	$\mathbf{local}\ x\ \mathbf{in}\ P$	Local Behavior
	$\mid$	$\mathbf{next}\ P$	Unit Delay
	$\mid$	$\mathbf{unless}\ c\ \mathbf{next}\ P$	Time-Out
	$\mid$	$\star\ P$	Asynchrony
	$\mid$	$!\ P$	Infinite Behavior

Example:

$$M \stackrel{\text{def}}{=} \; ! \; \textbf{when } \mathit{Go} = 1 \textbf{ do } \sum_{i \in \mathit{Notes}} \textbf{tell } \mathit{Note} = i \; \| \; \textbf{unless } \mathit{End} = 1 \textbf{ next } (\textbf{tell } \mathit{Go} = 1)$$

$$\mathit{Conductor} \stackrel{\text{def}}{=} \textbf{tell } \mathit{Go} = 1 \; \| \; \star(\textbf{tell } \mathit{End} = 1)$$

Disadvantages:

- No explicit metric notion of time is available.
- Time units are not homogeneous.
- There are no facilities to account for resource usage.
- Execution of processes is not preemptive.

The rtcc Calculus

Extends ntcc in these ways:

- Real-Time
- Delta Delay
- Strong Preemption
- Default Behaviour

The rtcc Calculus

Extends ntcc in these ways:

- Real-Time
- Delta Delay
- Strong Preemption
- Default Behaviour

Resources: A natural number indicating how many resources was provided by the environment.
Each process P takes some of these. When P is finished, it releases them.

The rtcc Calculus

Extends ntcc in these ways:

- Real-Time
- Delta Delay
- Strong Preemption
- Default Behaviour

Bounded Time: Each time unit is divided in a discrete sequence of minimal units called ticks.
The number of ticks will be the available time that processes have to execute.

The rtcc Calculus

Extends ntcc in these ways:

- Real-Time
- Delta Delay
- Strong Preemption
- Default Behaviour

Now a process can be delayed for at least certain number of ticks.

The rtcc Calculus

Extends ntcc in these ways:

- Real-Time
- Delta Delay
- Strong Preemption
- Default Behaviour

A process can be interrupted if a signal is given.

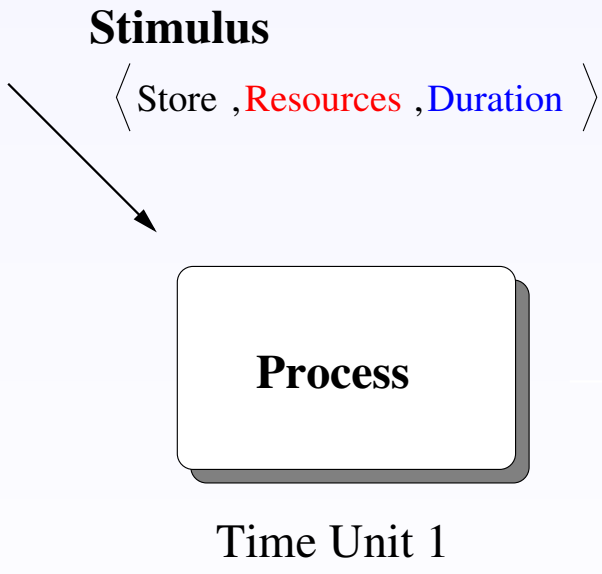
The rtcc Calculus

Extends ntcc in these ways:

- Real-Time
- Delta Delay
- Strong Preemption
- Default Behaviour

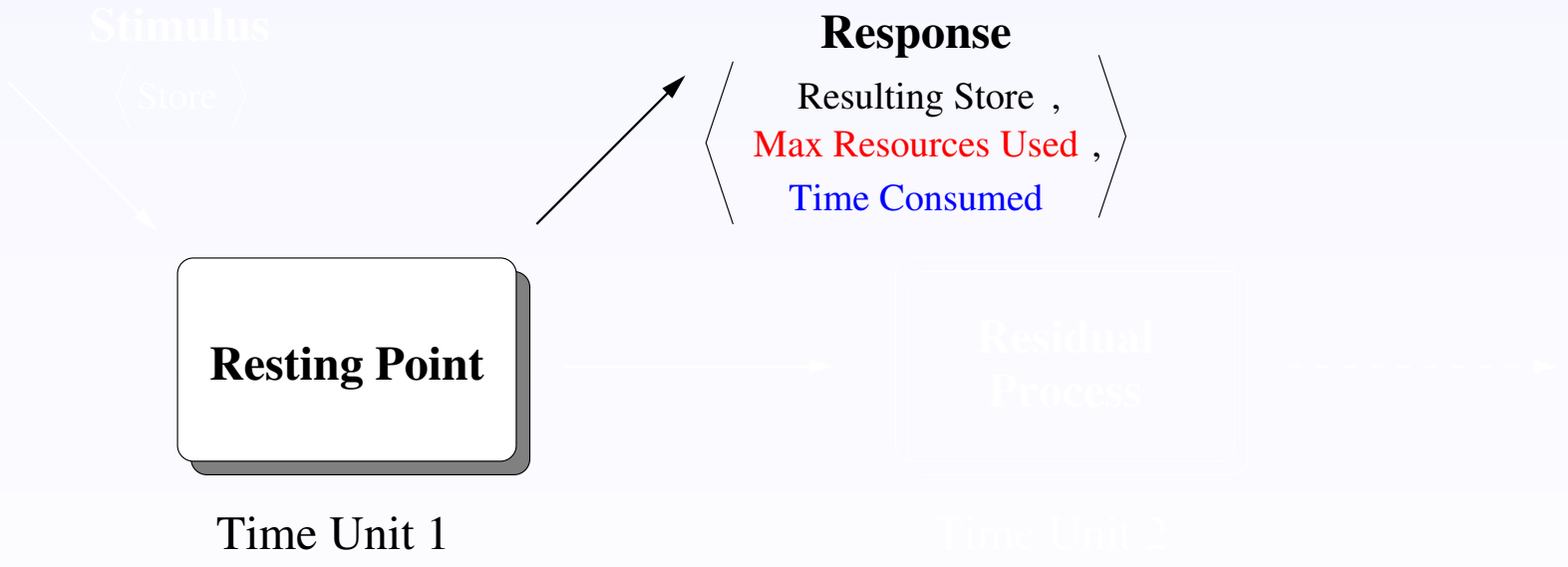
If the execution of a process is preempted then another process might be launched.

The rtcc Calculus

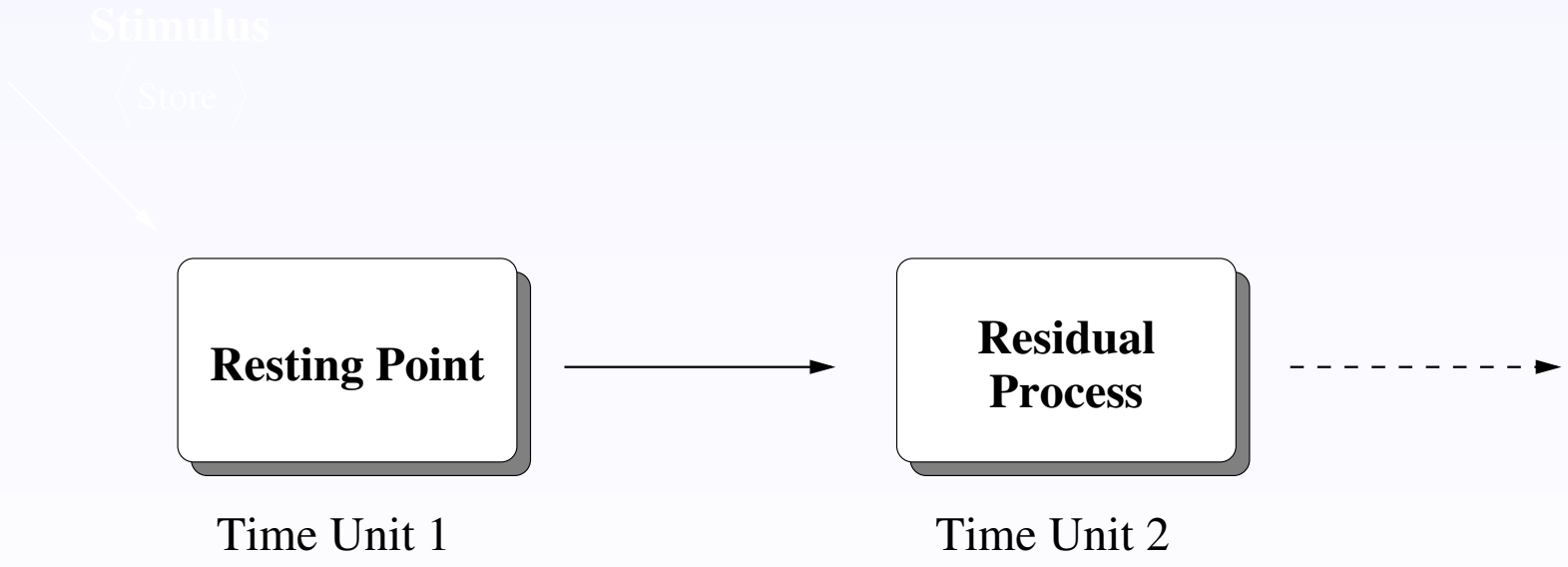


The rtcc Calculus

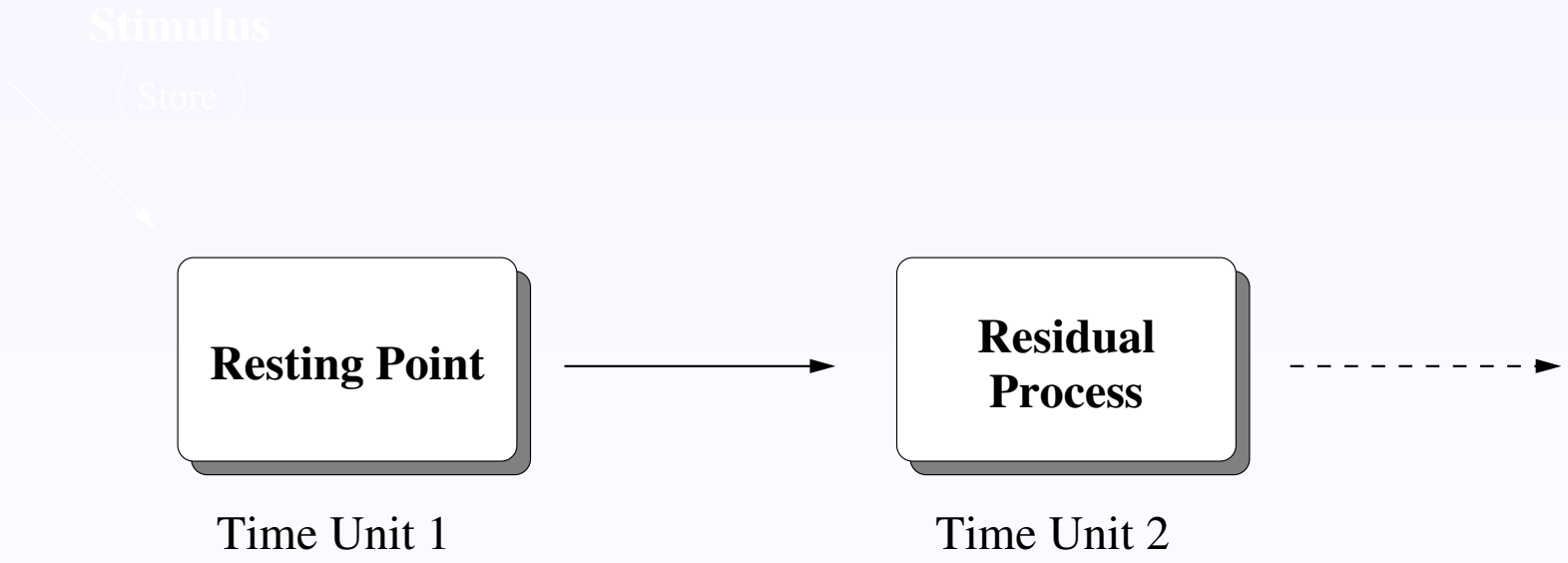
The rtcc Calculus



The rtcc Calculus



The rtcc Calculus



Stores are not automatically transferred from a time unit to the next one

Syntax

P, Q	$::=$	$\mathbf{tell}(c)$	Tell
	$\mid$	$\sum_{i \in I}$ **when** c_i **do** P_i	Nondeterminism
	$\mid$	$P \parallel Q$	Parallel Composition
	$\mid$	**local** x **in** P	Local Behavior
	$\mid$	**next** P	Unit Delay
	$\mid$	**delay** P **until** δ	Delta Delay
	$\mid$	**unless** c **next** P	Weak Time-Out
	$\mid$	**catch** c **in** P **finally** Q	Strong Time-Out
	$\mid$	$\star P$	Asynchrony
	$\mid$	**!** P	Replication

Operational Semantics

Operational semantics is based on configurations $\langle P, d, t \rangle$.

We have internal ($\xrightarrow{r}$) and observable transitions ($\overset{\alpha,\alpha'}{\Longrightarrow}$)

Operational Semantics

$$\frac{t - \Phi_T(c, d) \geq 0}{\langle \textbf{tell}(c), d, t \rangle \xrightarrow{1} \langle \textbf{skip}, d \wedge c, t - \Phi_T(c, d) \rangle}$$

Tell Operation

$\Phi_T(c, d)$: Time needed to post constraint c in store d.
The tell construct uses only 1 resource.

Operational Semantics

$$\frac{t - \Phi_A(c_j, d) \geq 0 \qquad d \vDash c_j, \qquad j \in I}{\langle \sum_{i \in I} \textbf{when } c_i \textbf{ do } P_i, d, t \rangle \xrightarrow{1} \langle P_j, d, t - \Phi_A(c_j, d) \rangle}$$

Ask Operation

$\Phi_A(c, d)$: Time needed to query if constraint c can be entailed by store d.

The ask construct uses only 1 resource.

Operational Semantics

$$\frac{\langle P, d, t\rangle \xrightarrow{s_p} \langle P', d'_p, t'_p\rangle \quad s_p \le r}{\langle P \parallel Q, d, t\rangle \xrightarrow{s_p} \langle P' \parallel Q, d'_p, t'_p\rangle} \qquad \frac{\langle Q, d, t\rangle \xrightarrow{s_q} \langle Q', d'_q, t'_q\rangle \quad s_q \le r}{\langle P \parallel Q, d, t\rangle \xrightarrow{s_q} \langle P \parallel Q', d'_q, t'_q\rangle}$$

$$\frac{\langle P, d, t\rangle \xrightarrow{s_p} \langle P', d'_p, t'_p\rangle \qquad \langle Q, d, t\rangle \xrightarrow{s_q} \langle Q', d'_q, t'_q\rangle \qquad s_p + s_q \le r}{\langle P \parallel Q, d, t\rangle \xrightarrow{s_p+s_q} \langle P' \parallel Q', d'_p \wedge d'_q, \min(t'_p, t'_q)\rangle}$$

Parallel Composition

Two above: Overlapping version

The one below: Truely parallel version

Operational Semantics

$$\frac{t - \Phi_A(c,d) \geq 0 \qquad d \vDash c}{\langle \textbf{catch } c \textbf{ in } P \textbf{ finally } Q, d, t\rangle \xrightarrow{1} \langle Q, d, t - \Phi_A(c,d)\rangle}$$

$$\frac{\langle P, d, t - \Phi_A(c,d)\rangle \xrightarrow{s} \langle P', d', t'\rangle \qquad d \nvDash c}{\langle \textbf{catch } c \textbf{ in } P \textbf{ finally } Q, d, t\rangle \xrightarrow{s} \langle \textbf{catch } c \textbf{ in } P' \textbf{ finally } Q, d', t'\rangle}$$

Strong Time-Out

$d \vDash c$: Process P is stopped and Q is launched.

$d \nvDash c$: Process P continues (if it can) its execution to P'.
Process P' is now guarded by c.

Operational Semantics

$$\frac{\delta \geq T - t \qquad t > 0}{\langle \textbf{delay } P \textbf{ until } \delta, d, t \rangle \xrightarrow{0} \langle \textbf{delay } P \textbf{ until } \delta, d, t - 1 \rangle}$$

$$\frac{\delta < T - t}{\langle \textbf{delay } P \textbf{ until } \delta, d, t \rangle \xrightarrow{0} \langle P, d, t \rangle}$$

Delta Delay Operation

T: Total duration of the time unit (given by the environment).
In each transition the available time is reduce one tick until the delay is greater than or equal to the current time.

Operational Semantics

$$\frac{\langle P, c, t\rangle \rightarrow_S^* \langle Q, d, t'\rangle \not\rightarrow}{P \xrightarrow{(\langle c,r,t\rangle,\ \langle d,\max(S),t-t'\rangle)} R} \quad \text{if } R \equiv F(Q)$$

Residual Process

$$F(Q) = \begin{cases} R & \text{if } Q = \textbf{next } R \text{ or} \\ & \quad Q = \textbf{unless } c \textbf{ next } R \\ F(Q_1) \parallel F(Q_2) & \text{if } Q = Q_1 \parallel Q_2 \\ \textbf{catch } c \textbf{ in } F(R) \textbf{ finally } S & \text{if } Q = \textbf{catch } c \textbf{ in } R \textbf{ finally } S \\ \textbf{local } x \textbf{ in } F(R) & \text{if } Q = \textbf{local } x, c \textbf{ in } R \\ \textbf{skip} & \text{Otherwise} \end{cases}$$

Operational Semantics

Observations made to Processes:

$$P = P_1 \xrightarrow{(\iota_1, o_1)} P_2 \xrightarrow{(\iota_2, o_2)} P_3 \xrightarrow{(\iota_3, o_3)} \ldots$$

denoted as $P \xrightarrow{(\alpha,\alpha')} \omega$ with $\alpha = \iota_1 \cdot \iota_2 \ldots$, $\alpha' = o_1 \cdot o_2 \ldots$

input-output behaviour of P

$$io(P) = \{(\alpha, \alpha') \mid P \xrightarrow{(\alpha,\alpha')} \omega\}$$

Denotational Semantics

A denotation is a labelled K-valued Chu space.

Chu Spaces

A matrix $\mathcal{C} = \langle A, X, \lambda \rangle$ over a set K:

- A: The events
- X: The states
- λ: The label
- K: The alphabet

Denotational Semantics

Events perform actions. States record event ocurrence.

Labels are functions $\lambda : A \to Act$, where each element of Act, the set of possible actions, is composed by:

- the actual information (constraints or time)
- a tag: $\begin{cases} \texttt{tell} & \text{posting information} \\ \texttt{ask} & \text{querying for information} \\ \texttt{time} & \text{time this event actually occurs} \end{cases}$

Denotational Semantics

The elements of K are the possible values of an event in a given state.

$$K = \begin{cases} 0 & \textit{before} & \text{it has not yet started} \\ \lrcorner\!\ulcorner & \textit{during} & \text{it is happening} \\ 1 & \textit{after} & \text{it is finishing} \\ \times & \textit{instead} & \text{it has been canceled} \end{cases}$$

Denotational Semantics

$$[\![\textbf{tell}(c)]\!] = \quad c \quad \boxed{0 \quad \lrcorner\!\ulcorner \quad 1} \qquad \lambda(c) \mapsto \langle c, \texttt{tell} \rangle$$

$$[\![\textbf{when } c \textbf{ do } P]\!] = [\![\textbf{tell}(c)]\!] \; ; \; [\![P]\!] \qquad \lambda(c) \mapsto \langle c, \texttt{ask} \rangle$$

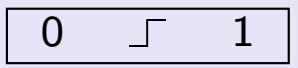
$$[\![P \parallel Q]\!] = [\![P]\!] \wedge [\![Q]\!]$$

$$[\![\textbf{catch } c \textbf{ in } P \textbf{ finally } Q]\!] = (\checkmark [\![\textbf{tell}(c)]\!] \wedge (\overline{A} \vee \not{A}) \wedge [\![Q]\!]) \vee$$
$$([\![\textbf{tell}(c)]\!] = 0 \wedge [\![P]\!] \wedge \overline{B})$$
$$\lambda(c) \mapsto \langle c, \texttt{ask} \rangle$$

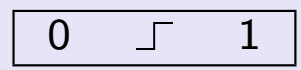
Denotational Semantics

$$[\![\textbf{delay } P \textbf{ until } \delta]\!] = DELAY \ ; \ \mathcal{P}$$

$$[\![\textbf{next } P]\!] = CLOCK \ ; \ \mathcal{P}$$

$DELAY$ = delay 

$\lambda(delay) = \langle \delta, \texttt{time} \rangle$

$CLOCK$ = clock 

$\lambda(clock) = \langle T, \texttt{time} \rangle$

Example 1:

Let $P \stackrel{\text{def}}{=}$ **tell**(a_1) and $Q \stackrel{\text{def}}{=}$ **when** a_2 **do tell**(a_3). Then

$$[\![P + Q]\!] = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{|ccccccc|} \hline 0 & \sqcap & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqcap & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & \sqcap & 1 \\ \hline \end{array}$$

$$\lambda(a_1) = \langle a_1, \texttt{tell} \rangle$$
$$\lambda(a_2) = \langle a_2, \texttt{ask} \rangle$$
$$\lambda(a_3) = \langle a_3, \texttt{tell} \rangle$$

Example 2:

Let $P \stackrel{\text{def}}{=}$ **catch** r **in tell**(b) and $Q \stackrel{\text{def}}{=}$ **tell**(b). Then:

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

r	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0
b_1	0	×	0	⌐	1	0	×	0	⌐	1	0	×	0	⌐	1
b_2	0	0	0	0	0	⌐	⌐	⌐	⌐	⌐	1	1	1	1	1

$\lambda(r) = \langle r, \texttt{ask} \rangle$
$\lambda(b_1) = \langle b, \texttt{tell} \rangle$
$\lambda(b_2) = \langle b, \texttt{tell} \rangle$

Denotational Semantics

Denotational Inputs:
Let $Ins : \mathbb{N} \to |D| \times \mathbb{N} \times \mathbb{N}$ be the function that given a time unit index, it will return a tuple consisting of a constraint representing the initial store, a number of resources and an amount of time, for that time unit. We denote $Ins(i).j$ the jth component ($j \in \{1, 2, 3\}$) of $Ins(i)$.

Denotational Semantics

Definition

- Step
- Run
- Valid Step
- Distance
- Valid Run

Denotational Outputs:
Let Ω be the set of valid runs R. Let $Outs : \Omega \to |D| \times \mathbb{N} \times \mathbb{N}$ be function which given a valid run of a Chu space will return a tuple consisting of a constraint representing a store, a number of resources and an amount of time (the outputs of a process).

Denotational Semantics

The observables of a process then, can be recovered as follows:

Denotational Observables:
Let $R_{P,i}$ be the set of valid runs of $[\![P]\!]_i$. The observations of a process P is the set of chains of its inputs and outputs, that is

$$Obs(P) = \{(\alpha, \alpha') \mid \alpha(i) = Ins(i) \text{ and } \alpha'(i) = Outs(R_{P,i})\}$$

Denotational Semantics

The observables of a process then, can be recovered as follows:

Denotational Observables:
Let $R_{P,i}$ be the set of valid runs of $[\![P]\!]_i$. The observations of a process P is the set of chains of its inputs and outputs, that is

$$Obs(P) = \{(\alpha, \alpha') \mid \alpha(i) = Ins(i) \text{ and } \alpha'(i) = Outs(R_{P,i})\}$$

Full Abstraction:
For two processes P and Q, $Obs(P) = Obs(Q)$ iff $io(P) = io(Q)$.

Real-time Logics

We define a Real-time logic for a simplify transition system (without resources): We assume Maximal Parallelism

Based on RTTL. An explicit clock variable T.

Syntax:

$$A, B, \ldots := c \mid \pi \mid \neg A \mid A \wedge A \mid \exists x.A \mid \Box\ A \mid \Diamond\ A \mid \circ A$$

Semantics:

Definition

A *timed observation sequence* $\rho = \langle \sigma, \tau \rangle$ is a pair consisting of an infinite sequence σ of states, and a monotonic function $\tau : \mathbb{N} \to \mathcal{I}$ that maps every $c_i \in \sigma$ to a time interval. A timed observation sequence is defined as

$$\langle c_1, \tau_1 \rangle \to \langle c_2, \tau_2 \rangle \to \langle c_3, \tau_3 \rangle \to \ldots$$

Real-time Logics

$$\frac{}{\textbf{tell}(c) \ \vdash \ c} \quad \text{if } \langle \rho, 1 \rangle \vDash T + \Phi_T(c, \sigma(1)) \leq r(\tau_1)$$

$$\frac{\forall i \ \in I \quad P_i \vdash A_i}{\sum_{i \in I} \textbf{when } c_i \textbf{ do } P_i \ \vdash \ \bigvee_{i \in I} (c_i \wedge A_i) \ \vee \ \bigwedge_{i \in I} (\neg c_i)} \quad \text{if } \langle \rho, 1 \rangle \vDash T + \Phi_A(c_i, \sigma(1)) \leq r(\tau_1)$$

$$\frac{P \ \vdash \ A \qquad Q \ \vdash \ B}{P \parallel Q \ \vdash A \wedge B} \qquad \frac{P \ \vdash \ A}{\textbf{local } x \textbf{ in } P \ \vdash \ \exists_x A}$$

$$\frac{P \ \vdash \ A}{\textbf{unless } c \textbf{ next } P \ \vdash \ c \vee A} \quad \text{if } \langle \rho, 1 \rangle \vDash T + \Phi_A(c, \sigma(1)) \leq r(\tau_1)$$

$$\frac{P \ \vdash \ A \qquad Q \ \vdash \ B}{\textbf{catch } c \textbf{ in } P \textbf{ finally } Q \ \vdash \ (\neg c \wedge A) \vee (c \wedge B)} \quad \text{if } \langle \rho, 1 \rangle \vDash T + \Phi_A(c, \sigma(1)) \leq r(\tau_1)$$

$$\frac{P \ \vdash \ A}{\textbf{delay } P \textbf{ until } \delta \ \vdash \ A} \quad \text{if } \langle \rho, 1 \rangle \vDash T \geq l(\tau_1) + \delta$$

$$\frac{P \ \vdash \ A}{\textbf{next } P \ \vdash \ A} \qquad \frac{P \ \vdash \ A}{!P \ \vdash \ \Box A} \qquad \frac{P \ \vdash \ A}{\star P \ \vdash \ \Diamond A}$$

Concluding Remarks

A survey of formal models for music. We review some of the formal models that have been proposed for expressing a variety of musical applications from a computation perspective.

Development of `rtcc`. We develop a new calculus justified in the lack of a formalism capable of expressing both reactive and real-time systems for music.

Applications. We illustrate the potential of the new calculus by formalizing a well-known musical tool and a musical improvisation process.

Future Work

- *Real-time logic with resources or Girard's Linear Logic*.
- *An interpreter*.
- *Modelling of complex musical systems*.
- *Expressivity proofs*
- *Native notion of "repair processes"*

Thank you!