SEMINARIO I

Gerardo M. Sarria M.

Pontificia Universidad Javeriana

Noviembre 1 de 2006

TIEMPO, CONCURRENCIA Y RESTRICCIONES EN LA MÚSICA

Motivación: Informática Musical

La composición musical y la improvisación son tareas complejas que implican definir y controlar actividades concurrentes.

Motivación: Informática Musical

La composición musical y la improvisación son tareas complejas que implican definir y controlar actividades concurrentes.

Algunos aspectos de la composición e improvisación:

- Actividades concurrentes (e.g. voces hablando en paralelo)
- Restricciones temporales (e.g. espacio entre notas)
- Sincronización (e.g. patrones de ritmo)

Motivación: Informática Musical

La composición musical y la improvisación son tareas complejas que implican definir y controlar actividades concurrentes.

Algunos aspectos de la composición e improvisación:

- Actividades concurrentes (e.g. voces hablando en paralelo)
- Restricciones temporales (e.g. espacio entre notas)
- Sincronización (e.g. patrones de ritmo)

Se requiere un modelo para sistemas temporales, concurrentes síncronos y asíncronos, y con restricciones que puedan ser usados como plataforma matemática para la composición y la improvisación.

Motivación: Cálculo de Procesos

Los Cálculos de Procesos son formalismos para modelar y analizar actividades concurrentes.

Motivación: Cálculo de Procesos

Los Cálculos de Procesos son formalismos para modelar y analizar actividades concurrentes.

Dichos cálculos tratan procesos de igual manera que el cálculo λ trata funciones calculables:

- Términos: E.g. $P \| Q$
- Reglas de Reducción: E.g.

$$\frac{P \rightarrow P'}{P \| Q \rightarrow P' \| Q}$$

Gerardo M.
Sarria M.

Cálculos

- λ&
- PiCO
- CC
- TCC
- Hybrid cc
- NTCC
- Ambient*
- CHU Spaces*

Cálculo $\lambda\&$

El cálculo $\lambda\&$ es una extensión del cálculo $\lambda_{\leq}$, el cual es el cálculo λ tipado con una relación de subtipos. Una descripción formal de $\lambda\&$ se puede resumir en las siguientes definiciones.

Cálculo $\lambda\&$

El cálculo $\lambda\&$ es una extensión del cálculo $\lambda_{\leq}$, el cual es el cálculo λ tipado con una relación de subtipos. Una descripción formal de $\lambda\&$ se puede resumir en las siguientes definiciones.

Términos

M	$::=$	x^V	variable
	$\mid$	$\lambda x^V.M$	función
	$\mid$	$M \cdot M$	aplicación de función
	$\mid$	$\langle \ell_1 = M, \ldots, \ell_n = M \rangle$	registros
	$\mid$	$M.\ell$	valor del un campo
	$\mid$	i^V	identificador de una ubicación de tipo `ref` V
	$\mid$	$\uparrow M$	el contenido de una ubicación M
	$\mid$	$M := M$	asignación
	$\mid$	$M; M$	secuencia
	$\mid$	nil	una constante de tipo ()
	$\mid$	ϵ	función de sobrecarga vacía
	$\mid$	$(M\&^V M)$	función de sobrecarga
	$\mid$	$M \bullet M$	aplicación de sobrecarga

donde V denota un tipo.

Gerardo M.
Sarria M.

Reducción

$$(\alpha) \qquad \lambda x^U.M = \lambda y^U.(M[x := y]) \qquad y \notin FV(M)$$

$$(\eta) \qquad \lambda x^U.Mx = M$$

$$(\beta) \qquad (\lambda x^U.M)N \ \rhd \ M[x^U := N]$$

$(\beta_\&)$ if $N : U$ is closed and in normal form, and $U_j = min_{i=1\ldots n}\{U_i | U \leq U_i\}$, then

$$(M_1 \&^{\{U_i \to V_i\}_{i=1\ldots n}} M_2) \bullet N \ \rhd \ \begin{cases} M_1 \bullet N & \text{for } j < n \\ M_2 \ N & \text{for } j = n \end{cases}$$

donde $FV(M)$ es el conjunto de las variables libres del término M.

Gerardo M.
Sarria M.

Cálculo PiCO

El cálculo π^+ extiende el cálculo π con la noción de restricciones.

Cálculo PiCO

El cálculo π^+ extiende el cálculo π con la noción de restricciones.

En PiCO se adiciona a π^+ la noción de objetos y paso de mensajes sincronizados por restricciones.

Cálculo PiCO

El cálculo π^+ extiende el cálculo π con la noción de restricciones.

En PiCO se adiciona a π^+ la noción de objetos y paso de mensajes sincronizados por restricciones.

Las restricciones y los objetos concurrentes son nociones primitivas al nivel del cálculo.

Cálculo PiCO

La sintaxis de PiCO es la siguiente:

Normal Processes: N	::=	O	Inaction or null process
	|	$I \lhd m$ **then** P	Message sent by I
	|	$(\phi_{\textbf{sender}}, \delta_{\textbf{forward}}) \rhd M$	Object with delegation condition $\delta_{\textbf{forward}}$ guarded by constraint ϕ
Constraint processes: R	::=	**tell** ϕ **then** P	*tell* process
	|	**ask** ϕ **then** P	*ask* process
Processes: P, Q	::=	**local** x **in** P	New variables x in P
	|	**local** a **in** P	New name a in P
	|	N	Normal process
	|	$P \mid Q$	Composition
	|	**clone** P	Replicated process
	|	R	Constraint process
Object identifiers: I, J, K	::=	a, I	Names
	|	v	Value
	|	x	Variable
Collection of Methods: M	::=	$[l_1 : (\widetilde{x_1})P_1 \ \& \ \ldots \ \& \ l_m : (\widetilde{x_m})P_m]$	
Messages: m	::=	$I : [\widetilde{I}]$	

Cálculo PiCO

Reducción

$$\frac{S \vdash_{\triangle} \phi[I' \ \textbf{sender}] \qquad |\widetilde{K}| = |\widetilde{x}|}{\langle I' \lhd I : [\widetilde{K}] \ \textbf{then} \ Q \mid (\phi_{\textbf{sender}}, \delta_{\textbf{forward}}) \rhd [I : (\widetilde{x})P \& \ldots]; S\rangle \rightarrow \langle Q \mid P\{\widetilde{K}/\widetilde{x'}, I'/\textbf{sender}\}; S\rangle}$$

$$\frac{S \vdash_{\triangle} \phi[I'/\textbf{sender}] \qquad S \sqcup \delta[I'/\textbf{forward}] \vdash_{\triangle} \bot \qquad I \notin \mathrm{Labels}(M)}{\left\langle \begin{pmatrix} I' \lhd I{:}[\widetilde{K}] \ \textbf{then} \ Q \mid \\ (\phi_{\textbf{sender}}, \delta_{\textbf{forward}}) \rhd M \end{pmatrix}; S \right\rangle \rightarrow \left\langle \begin{pmatrix} \textbf{local} \ J \ \textbf{in} \ \textbf{tell} \ \delta[J/\textbf{forward}] \ \textbf{then} \ (J \lhd I{:}[\widetilde{K}] \ \textbf{then} \ Q) \mid \\ (\phi_{\textbf{sender}}, \delta_{\textbf{forward}}) \rhd M \end{pmatrix}; S \right\rangle}$$

$$\langle \textbf{tell} \ \phi \ \textbf{then} \ P; S\rangle \rightarrow \langle P; S \wedge \phi\rangle$$

$$\frac{S \vdash_{\triangle} \phi}{\langle \textbf{ask} \ \phi \ \textbf{then} \ P; S\rangle \rightarrow \langle P; S\rangle} \quad , \quad \frac{S \vdash_{\triangle} \neg\phi}{\langle \textbf{ask} \ \phi \ \textbf{then} \ P; S\rangle \rightarrow \langle O; S\rangle}$$

$$\frac{\langle P; S\rangle \rightarrow \langle P'; S'\rangle}{\langle Q \mid P; S\rangle \rightarrow \langle Q \mid P'; S'\rangle}$$

$$\frac{x \notin fv(S), \qquad \langle P; S \gg \{x\}\rangle \rightarrow \langle P'; S'\rangle}{\langle \textbf{local} \ x \ \textbf{in} \ P; S\rangle \rightarrow \langle P'; S'\rangle}$$

$$\frac{a \notin fv(S), \qquad \langle P; S \gg \{a\}\rangle \rightarrow \langle P'; S'\rangle}{\langle \textbf{local} \ a \ \textbf{in} \ P; S\rangle \rightarrow \langle P'; S'\rangle}$$

$$\frac{\langle P_1; S_1\rangle \equiv_P \langle P_1'; S_1'\rangle \qquad \langle P_2; S_2\rangle \equiv_P \langle P_2'; S_2'\rangle \qquad \langle P_1; S_1\rangle \rightarrow \langle P_2; S_2\rangle}{\langle P_1'; S_1'\rangle \rightarrow \langle P_2'; S_2'\rangle}$$

Gerardo M.
Sarria M.

Concurrent Constraint Programming

Concurrent constraint programming (CC) es un modelo para especificar sistemas concurrentes en términos de restricciones.

Concurrent Constraint Programming

Concurrent constraint programming (CC) es un modelo para especificar sistemas concurrentes en términos de restricciones.

Una *restricción* es una fórmula de primer orden que representa información parcial sobre variables compartidas del sistema.

Concurrent Constraint Programming

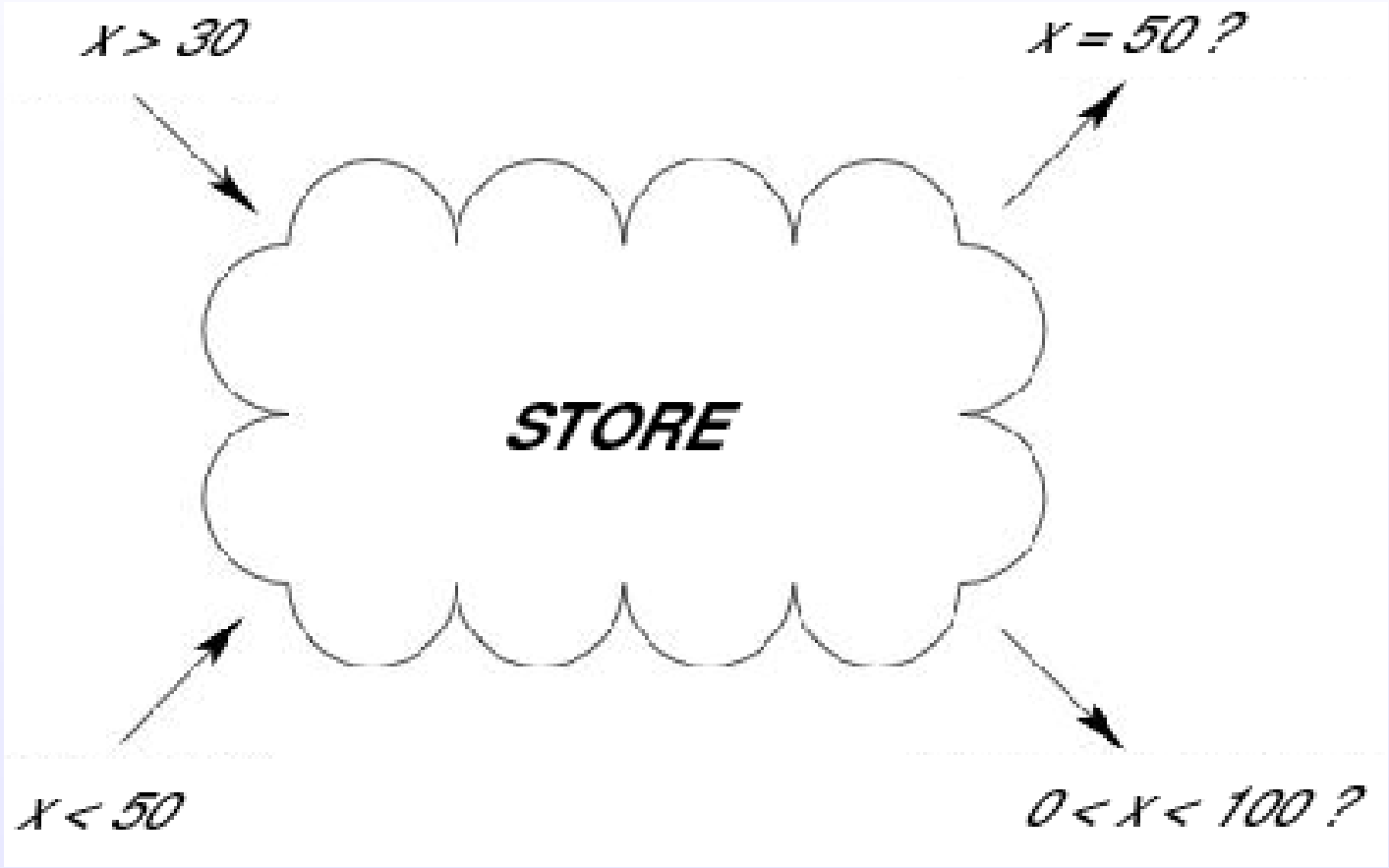
Concurrent constraint programming (CC) es un modelo para especificar sistemas concurrentes en términos de restricciones.

Una *restricción* es una fórmula de primer orden que representa información parcial sobre variables compartidas del sistema.

La información sobre las variables recide en un *store*, el cual es na conjunción de todas las restricciones aplicadas a las variables. Este *store* puede ser accesado por *agentes* (procesos que interactúan con el *store*) con dos operaciones básicas: `ask` y `tell`.

Gerardo M.
Sarria M.

Concurrent Constraint Programming



Gerardo M.
Sarria M.

Concurrent Constraint Programming

Un *sistema de restricciones* es un dominio donde las restricciones son parametrizadas.

Concurrent Constraint Programming

Un *sistema de restricciones* es un dominio donde las restricciones son parametrizadas.

Consiste:

- Una signatura Σ – i.e. un conjunto de funciones, constantes y símbolos de predicado con igualdad
- Una teoría Δ – i.e. un conjunto de sentencias sobre Σ con un modelo

Concurrent Constraint Programming

Un *sistema de restricciones* es un dominio donde las restricciones son parametrizadas.

Consiste:

- Una signatura Σ – i.e. un conjunto de funciones, constantes y símbolos de predicado con igualdad
- Una teoría Δ – i.e. un conjunto de sentencias sobre Σ con un modelo

Por ejemplo, el sistema de restricciones de conjuntos finitos puede tener una signatura que incluya los símbolos $\{0, succ, +, \times, =\}$ y cuya teoría es el conjunto de sentencias válidas en aritmética.

TCC

Una extensión de cc es el cálculo tcc, el cual ha sido creado para modelar y programar sistemas reactivos temporales.

TCC

Una extensión de cc es el cálculo tcc, el cual ha sido creado para modelar y programar sistemas reactivos temporales.

Este modelo extiende cc con las operaciones *delay* y *time-out*.

TCC

Una extensión de cc es el cálculo tcc, el cual ha sido creado para modelar y programar sistemas reactivos temporales.

Este modelo extiende cc con las operaciones *delay* y *time-out*.

En sistemas reactivos el tiempo está dividido en intervalos discretos y en cada intervalo un proceso recibe un estímulo (restricciones) del ambiente y produce una respuesta:

$$P_1 \xrightarrow{(c_1,d_1)} P_2 \xrightarrow{(c_2,d_2)} \dots P_i \xrightarrow{(c_i,d_i)} P_{i+1} \xrightarrow{(c_{i+1},d_{i+1})} \dots$$

TCC

Una extensión de cc es el cálculo tcc, el cual ha sido creado para modelar y programar sistemas reactivos temporales.

Este modelo extiende cc con las operaciones *delay* y *time-out*.

En sistemas reactivos el tiempo está dividido en intervalos discretos y en cada intervalo un proceso recibe un estímulo (restricciones) del ambiente y produce una respuesta:

$$P_1 \xrightarrow{(c_1,d_1)} P_2 \xrightarrow{(c_2,d_2)} \dots P_i \xrightarrow{(c_i,d_i)} P_{i+1} \xrightarrow{(c_{i+1},d_{i+1})} \dots$$

El tiempo es un conjunto de puntos que actúan como marcadores que distinguen un momento del siguiente.

TCC

The syntax of this model can be summarize as follows:

A	$::=$	c	Tell
	$\mid$	**now** c **then** A	Timed Positive Ask
	$\mid$	**now** c **else** A	Timed Negative Ask
	$\mid$	**next** A	Unit Delay
	$\mid$	**abort**	Abort
	$\mid$	**skip**	Skip
	$\mid$	$A \parallel A$	Parallel Composition
	$\mid$	X^A	Hiding
	$\mid$	P	Nested Program
	$\mid$	g	Procedure Call
g	$::=$	$p(t_1, \ldots, t_n)$	Procedure
P	$::=$	$\{D.A\}$	Program
D	$::=$	$g :: A$	Definition
	$\mid$	$D.D$	Conjuntion

Gerardo M.
Sarria M.

TCC

La semántica operacional está dada en términos de configuraciones.

TCC

La semántica operacional está dada en términos de configuraciones.

Una configuración es un conjunto Γ de agentes.

TCC

La semántica operacional está dada en términos de configuraciones.

Una configuración es un conjunto Γ de agentes.

Existen dos relaciones de transición $\longrightarrow$, $\rightsquigarrow$ sobre las configuraciones. $\longrightarrow$ representa transiciones *dentro* de un instante de tiempo, mientras $\rightsquigarrow$ representa una transición de un instante de tiempo al siguiente.

Reducción

$$
\begin{array}{rcll}
(\Gamma, \textbf{skip}) & \longrightarrow & \Gamma & \\
(\Gamma, \textbf{abort}) & \longrightarrow & \textbf{abort} & \\
(\Gamma, A \parallel B) & \longrightarrow & (\Gamma, A, B) & \\
(\Gamma, X^A) & \longrightarrow & (\Gamma, A[Y/X]) & (Y \text{ not free in } \Gamma) \\
(\Gamma, p(t_1, \ldots, t_n)) & \longrightarrow & (\Gamma, A[X_1 \mapsto t_1, \ldots, X_n \mapsto t_n]) &
\end{array}
$$

$$\frac{\sigma(\Gamma) \vdash c}{(\Gamma, \textbf{now } c \textbf{ then } A) \longrightarrow (\Gamma, A)}$$

$$\frac{\sigma(\Gamma) \vdash c}{(\Gamma, \textbf{now } c \textbf{ else } B) \longrightarrow \Gamma}$$

$$\frac{\Delta, \{\textbf{now } c_i \textbf{ else } A_i \mid i < n\} \not\longrightarrow}{\Delta, \{\textbf{now } c_i \textbf{ else } A_i \mid i < n\}, \{\textbf{next } B_j \mid j < m\}, \rightsquigarrow \{A_i \mid i < n\}, \{B_j \mid j < m\}}$$

donde $\sigma(\Gamma)$ representa el *store* y Δ el rango de los conjunto de agentes que no contienen *Timed Negative Asks* o *Unit Delays*.

Hybrid cc

Otra extensión de `cc` es `Default cc` la cual permite procesos no monotónicos y la expresión de procesos *por defecto*, es decir, información negativa (i.e. en el caso en que no sea posible deducir algo del *store* entonces algunos procesos serán ejecutados).

Hybrid cc

Otra extensión de `cc` es `Default cc` la cual permite procesos no monotónicos y la expresión de procesos *por defecto*, es decir, información negativa (i.e. en el caso en que no sea posible deducir algo del *store* entonces algunos procesos serán ejecutados).

La operación **if** a **else** A es permitida en este modelo. Ella impone la restricción A a menos que el resto del sistema imponga la restricción a.

Gerardo M. Sarria M.

Hybrid cc

`Time Default cc` extiende `Default cc` en un tiempo discreto.

Hybrid cc

`Time Default cc` extiende `Default cc` en un tiempo discreto.

Como `cc` no tiene el concepto de tiempo, éste es dividido en *ticks* discretos (como en `tcc`).

Hybrid cc

`Time Default cc` extiende `Default cc` en un tiempo discreto.

Como `cc` no tiene el concepto de tiempo, éste es dividido en *ticks* discretos (como en `tcc`).

De esta manera, **next** A (A será ejecutado en el siguiente tick) y **always** A (A será ejecutado por siempre) son adicionados al modelo.

Hybrid cc

`Time Default cc` extiende `Default cc` en un tiempo discreto.

Como `cc` no tiene el concepto de tiempo, éste es dividido en *ticks* discretos (como en `tcc`).

De esta manera, **next** A (A será ejecutado en el siguiente tick) y **always** A (A será ejecutado por siempre) son adicionados al modelo.

Todas las reglas de transición que definen la semántica operacional en `Default cc` son las mismas que en `cc`. La regla faltante (que corresponde al *negative ask*) es la siguiente:

$$\frac{a \not\vdash b}{\Gamma, \textbf{if } b \textbf{ else } B \ \rightarrow_a \ \Gamma, B}$$

Hybrid cc

Hybrid concurrent constraint programming es una extensión de `Default cc` sobre tiempo continuo.

Hybrid cc

Hybrid concurrent constraint programming es una extensión de `Default cc` sobre tiempo continuo.

Un solo constructor temporal, llamado **hence**, es agregado a las operaciones de `Default cc`. **hence** A impone la restricción A en todo instante de tiempo después del actual.

Hybrid cc

Hybrid concurrent constraint programming es una extensión de `Default cc` sobre tiempo continuo.

Un solo constructor temporal, llamado **hence**, es agregado a las operaciones de `Default cc`. **hence** A impone la restricción A en todo instante de tiempo después del actual.

A	::=	a	Tell
	|	**if** a **then** A	Positive Ask
	|	**if** a **else** A	Negative Ask
	|	**new** X **in** A	Hiding
	|	(A, B)	Parallel Composition
	|	**hence** A	Unconditional persistence

Hybrid cc

Reducción

$$(a, (\Gamma, \textbf{hence} A), \Delta) \xrightarrow{\text{hcc}}_{b,r} (a, (\Gamma, A), (\Delta, A, \textbf{hence} A))$$

$$(a, (\Gamma, (\mathrm{A}, \mathrm{B})), \Delta) \xrightarrow{\text{hcc}}_{b,r} (a, (\Gamma, A, B), \Delta)$$

$$(a, (\Gamma, \textbf{new } X \textbf{ in } A), \Delta) \xrightarrow{\text{hcc}}_{b,r} (a, (\Gamma, A[Y/X]), \Delta) \quad (Y \text{ not free in } a, A, \Delta, \Gamma)$$

$$\frac{a, \mathtt{cont}(\sigma(\Gamma)) \vdash^r a'}{(a, (\Gamma, \textbf{if } a' \textbf{ then } B), \Delta) \xrightarrow{\text{hcc}}_{b,r} (a, (\Gamma, B), \Delta)}$$

$$\frac{a, \mathtt{cont}(b) \nvdash^r a'}{(a, (\Gamma, \textbf{if } a' \textbf{ else } B), \Delta) \xrightarrow{\text{hcc}}_{b,r} (a, (\Gamma, B), \Delta)}$$

donde $\mathtt{cont}(b)$ es un *token* que si se mantiene en el tiempo $t \in \mathbb{R}^+$, significa que b está en todo tiempo $r > t$.

Gerardo M.
Sarria M.

NTCC

El cálculo NTCC extiende `tcc` con las nociones de asincronía y no determinismo.

NTCC

El cálculo NTCC extiende tcc con las nociones de asincronía y no determinismo.

Sintaxis

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

donde c es una restricción y x es una variable.

Reducción

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

$$\frac{}{\langle \sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i, d\rangle \longrightarrow \langle P_j, d\rangle}\ \text{if } d \models c_j, j \in I$$

$$\frac{\langle P, c\rangle \longrightarrow \langle P', d\rangle}{\langle P \| Q, c\rangle \longrightarrow \langle P' \| Q, d\rangle}$$

$$\frac{}{\langle \mathbf{unless}\ c\ \mathbf{next}\ P, d\rangle \longrightarrow \langle \mathbf{skip}, d\rangle}\ \text{if } d \models c$$

$$\frac{\langle P, c \wedge \exists_x d\rangle \longrightarrow \langle P', c'\rangle}{\langle (\mathbf{local}\ x, c)P, d\rangle \longrightarrow \langle (\mathbf{local}\ x, c')P', d \wedge \exists_x c'\rangle}$$

$$\frac{}{\langle \star P, d\rangle \longrightarrow \langle \mathbf{next}^n\ P, d\rangle}\ \text{if } n \geq 0$$

$$\frac{}{\langle !P, d\rangle \longrightarrow \langle P \|\ \mathbf{next}\ !P, d\rangle}$$

$$\frac{\gamma_1 \longrightarrow \gamma_2}{\gamma_1' \longrightarrow \gamma_2'}\ \text{if } \gamma_1 \equiv \gamma_2 \text{ and } \gamma_1' \equiv \gamma_2'$$

Ejemplo Musical

Improvisación simple controlada (y sincronizada):

$$M \stackrel{\text{def}}{=} \; !(\textbf{when } (go = 1) \textbf{ do } (\textbf{tell } (note = 60) + \textbf{tell}(note = 64))) \parallel \textbf{unless } end = 1 \textbf{ next } \textbf{tell}(go = 1)$$

$$C \stackrel{\text{def}}{=} \; \textbf{tell}(go = 1) \parallel \star \; ! \; \textbf{tell}(end = 1)$$

$$\text{init} \stackrel{\text{def}}{=} \; M \parallel C$$

Ejemplo Musical

Improvisación simple controlada (y sincronizada):

$$M \stackrel{\text{def}}{=} \textbf{!}(\textbf{when } (go = 1) \textbf{ do } (\textbf{tell } (note = 60) + \textbf{tell}(note = 64))) \parallel \textbf{unless } end = 1 \textbf{ next } \textbf{tell}(go = 1)$$

$$C \stackrel{\text{def}}{=} \textbf{tell}(go = 1) \parallel \star \textbf{ ! } \textbf{tell}(end = 1)$$

$$\text{init} \stackrel{\text{def}}{=} M \parallel C$$

Se pueden modelar sistemas de improvisación más complejos con n músicos y un conductor de esta manera:

$$\text{init} \stackrel{\text{def}}{=} M_1 \parallel M_2 \parallel \ldots \parallel M_n \parallel C$$

Gerardo M.
Sarria M.

Problemas

- Tiempo Real

Problemas

- Tiempo Real
- Procesos rítmicos (patrones)

Problemas

- Tiempo Real
- Procesos rítmicos (patrones)
- Modulaciones métricas

Problemas

- Tiempo Real
- Procesos rítmicos (patrones)
- Modulaciones métricas
- Modelos probabilísticos (solucionado con SNTCC)

Problemas

- Tiempo Real
- Procesos rítmicos (patrones)
- Modulaciones métricas
- Modelos probabilísticos (solucionado con SNTCC)
- Lenguajes de programación

Problemas

- Tiempo Real
- Procesos rítmicos (patrones)
- Modulaciones métricas
- Modelos probabilísticos (solucionado con SNTCC)
- Lenguajes de programación
- Máquinas abstractas de los cálculos

Fin de la Presentación