

(A Simple)
Domain Theory for Concurrency

PhD Defence
November 21st 2003
Mikkel Nygaard

Department of Computer Science
University of Aarhus

Joint work with



(Oliva'99)

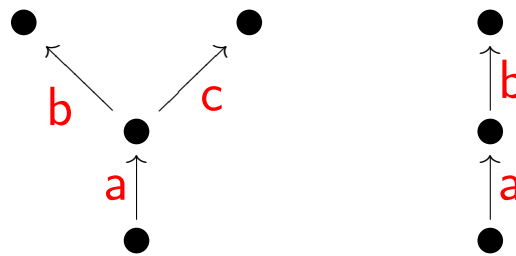
Outline

- a taste of concurrency theory
 - models
 - languages
 - equivalences
- a simple domain theory for concurrency
 - path semantics
 - HOPLA
 - relating semantics
- extensions
 - strong correspondence
 - Affine HOPLA
 - related and future work

Process models



Tree model:



(Milner'80)

Trace model:

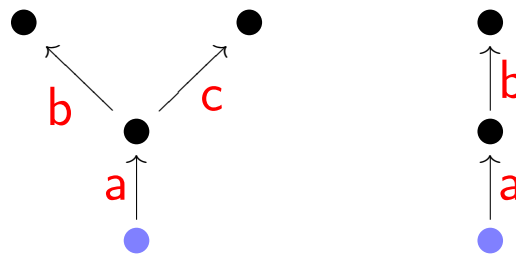


(Hoare'81)

Process models



Tree model:



Trace model:

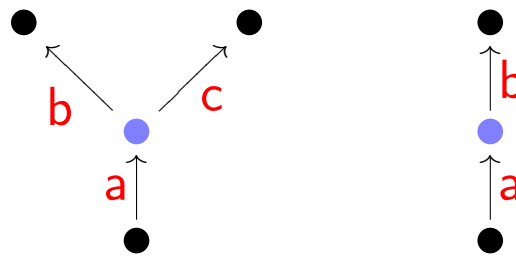
$\{a, ab, ac\}$

$\{a, ab\}$

Process models



Tree model:



Trace model:

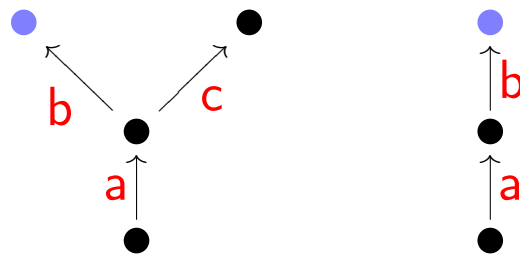
$\{a, ab, ac\}$

$\{a, ab\}$

Process models



Tree model:



Trace model:

$\{a, ab, ac\}$ $\{a, ab\}$

Milner's CCS

- CCS = Calculus of Communicating Systems (Milner'80)
- syntax $t, u ::= x \mid \text{rec } x.t \mid \sum_{i \in I} t_i \mid \alpha.t \mid t|u \mid t \setminus S \mid t[f]$
- recursion, nondeterministic sum, prefixing

I **a** accept money
then give either
b beer or **c** coffee
(then restart)



$a.(b.\emptyset + c.\emptyset)$
 $\text{rec } x.a.(b.x + c.x)$

- parallel composition



Operational semantics for CCS

$$\frac{t[\text{rec } x.t/x] \xrightarrow{\alpha} t'}{\text{rec } x.t \xrightarrow{\alpha} t'} \quad \frac{t_j \xrightarrow{\alpha} t'}{\sum_{i \in I} t_i \xrightarrow{\alpha} t'} \quad j \in I \quad \frac{}{\alpha.t \xrightarrow{\alpha} t}$$

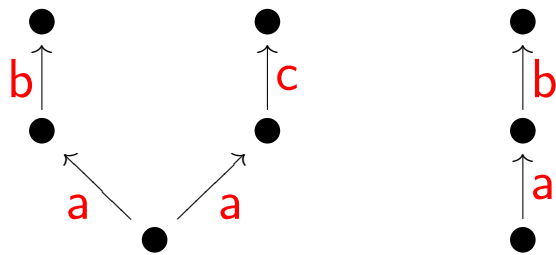
$$\frac{t \xrightarrow{\alpha} t'}{t|u \xrightarrow{\alpha} t'|u} \quad \frac{t \xrightarrow{l} t' \quad u \xrightarrow{\bar{l}} u'}{t|u \xrightarrow{\tau} t'|u'} \quad \frac{u \xrightarrow{\alpha} u'}{t|u \xrightarrow{\alpha} t|u'}$$

$$\frac{t \xrightarrow{\alpha} t'}{t \setminus S \xrightarrow{\alpha} t' \setminus S} \quad \alpha \notin S \quad \frac{t \xrightarrow{\alpha} t'}{t[f] \xrightarrow{f(\alpha)} t'[f]}$$

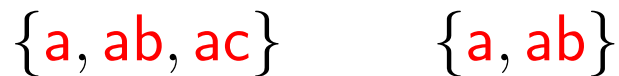
Another vending machine



Tree model:



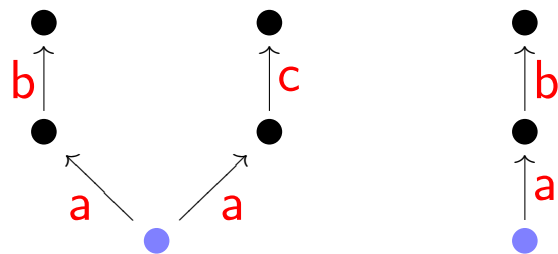
Trace model:



Another vending machine



Tree model:



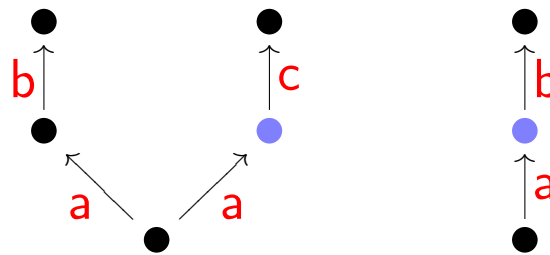
Trace model:



Another vending machine



Tree model:

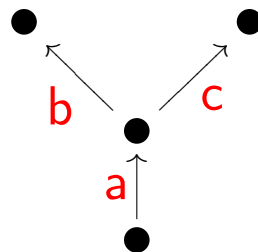


Trace model:

$\{a, ab, ac\}$ $\{a, ab\}$

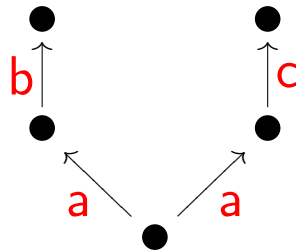
Trace equivalence and simulation

I **a**cept money
then give either
beer or **c**offee



$\{a, ab, ac\}$

I either
acept money
then give **b**eer
or
acept money
then give **c**offee



$\{a, ab, ac\}$

- *trace equivalence* = same trace set (Hoare'81)
- *bisimilarity* = related by 2-way simulation (Park'81, Milner'89)

Bisimulation and CCS

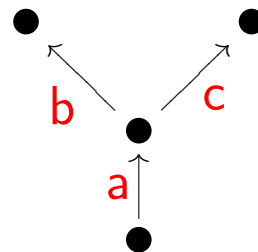
- bisimilarity is a *congruence* for CCS

$$t_1 \sim t_2 \implies \forall C. C(t_1) \sim C(t_2)$$

- Hennessy-Milner logic

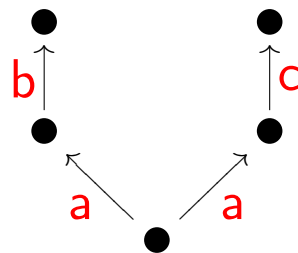
$$\phi ::= \langle \alpha \rangle \phi \mid \bigwedge_{i \in I} \phi_i \mid \neg \phi$$

I **a** accept money
then give either
beer or **c**offee



$$\langle a \rangle (\langle b \rangle T \wedge \langle c \rangle T)$$

I either
a accept money
then give **b**eer
or
a accept money
then give **c**offee



$$\neg \langle a \rangle (\langle b \rangle T \wedge \langle c \rangle T)$$

A fragmented picture

- models
 - trace sets, synchronisation trees, transition systems, . . .
 - Mazurkiewicz trace sets, event structures, Petri nets, strand spaces, nondeterministic dataflow, . . .
- languages
 - CCS, CSP, SCCS, CHOCS, . . .
 - π -calculus, higher-order π -calculus, ambient calculus, . . .
- equivalences
 - bisimulation, history-preserving bisimulation, barbed bisimulation, . . .
 - trace, simulation, contextual, failure, . . .

Domain theory for concurrency

- benefits
 - a global mathematical world of process models
 - guidance in defining and relating process languages
 - guidance in defining and relating process equivalences
 - methods of reasoning, logics, unified theory
- requirements
 - higher-order processes
 - independence models
 - name-generation
 - operational interpretation

Towards a domain theory

- categorical structure
 - Models for concurrency (Winskel&Nielsen'95)
 - Bisimulation from open maps (Joyal&Nielsen&Winskel'96)
- presheaf models (generalised tree)
 - PhD theses: Cattani 1999, Hildebrandt 2000
 - CCS-like languages, models, equivalences
(Winskel'96, Cattani&Winskel'96, Fiore&Cattani&Winskel'99, Hildebrandt'99)
 - higher-order processes (Winskel'98)
 - independence (Hildebrandt&Panangaden&Winskel'98, Winskel'99)
 - name-generation (Cattani&Stark&Winskel'97)
- operational interpretation?

Outline

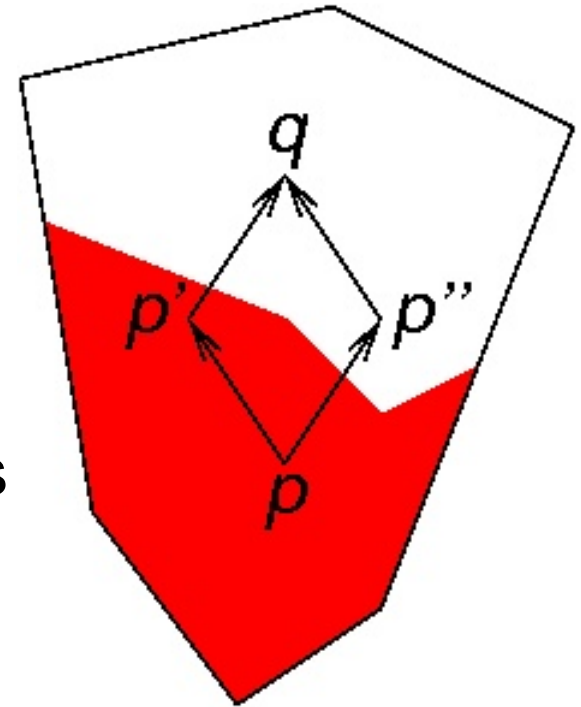
- a taste of concurrency theory
 - models
 - languages
 - equivalences
- a simple domain theory for concurrency
 - path semantics
 - HOPLA
 - relating semantics
- extensions
 - strong correspondence
 - Affine HOPLA
 - related and future work

Path semantics

- idea from presheaf models and Hennessy's work (Hennessy'94)

process =
collection of computation paths

- here “collection” means “set”
- path orders \mathbb{P} , paths and extensions
- process of type $\mathbb{P} = \text{path set } X \subseteq \mathbb{P}$
- *path equivalence* = equality of path sets
- nondeterministic sum = union ($\sum_{i \in I} X_i$)
- inactive process = empty set (\emptyset)
- nondeterministic domain
- process operations?



(Hennessy&Plotkin'79)

Mathematical structure

Lin

path orders
join-preserving maps

symmetric
monoidal
adjunction

Cts

path orders
continuous maps

linear logic:

$\mathbb{O}, \mathbb{P} \& \mathbb{Q}$

$\mathbb{P} \times \mathbb{Q}$

$\mathbb{P} \multimap \mathbb{Q}$

$\sum_{\alpha \in A} \mathbb{P}_\alpha$

$!\mathbb{P}$

intuitionistic logic:

$\mathbb{O}, \mathbb{P} \& \mathbb{Q}$

$\mathbb{P} \rightarrow \mathbb{Q} = !\mathbb{P} \multimap \mathbb{Q}$

(Seely'87, Benton'94)

(Cattani&Winskel'03)

HOPLA—a Higher-Order Process Language

- language constructs

(all types)

$\mathbb{P} \rightarrow \mathbb{Q}$

$\Sigma_{\alpha \in A} \mathbb{P}_\alpha$

$!\mathbb{P}$

$\mu_j \vec{T}. \vec{T}$

$rec\ x.t \quad \Sigma_{i \in I} t_i$

$\lambda x.t \quad t\ u$

$\alpha t \quad \pi_\alpha t$

$!t \quad [u > !x \Rightarrow t]$

$abs\ t \quad rep\ t$

- CCS prefixing = injection + anonymous prefixing

$\alpha.t \rightsquigarrow \alpha!t$

Operational semantics for HOPLA

Actions: $a ::= u \mapsto a \mid \alpha a \mid ! \mid \text{abs } a$

$$\frac{t[\text{rec } x.t/x] \xrightarrow{a} t'}{\text{rec } x.t \xrightarrow{a} t'} \quad \frac{t_j \xrightarrow{a} t'}{\sum_{i \in I} t_i \xrightarrow{a} t'} \quad j \in I$$

$$\frac{t[u/x] \xrightarrow{a} t'}{\lambda x.t \xrightarrow{u \mapsto a} t'}$$

$$\frac{t \xrightarrow{u \mapsto a} t'}{t u \xrightarrow{a} t'}$$

$$\frac{t \xrightarrow{a} t'}{\alpha t \xrightarrow{\alpha a} t'}$$

$$\frac{t \xrightarrow{\alpha a} t'}{\pi_{\alpha} t \xrightarrow{a} t'}$$

$$\frac{}{!t \xrightarrow{!} t} \quad \frac{u \xrightarrow{!} u' \quad t[u'/x] \xrightarrow{a} t'}{[u > !x \Rightarrow t] \xrightarrow{a} t'}$$

$$\frac{t \xrightarrow{a} t'}{\text{abs } t \xrightarrow{\text{abs } a} t'}$$

$$\frac{t \xrightarrow{\text{abs } a} t'}{\text{rep } t \xrightarrow{a} t'}$$

Bisimulation

Bisimilar terms:

$$\text{rec } x.t \sim t[\text{rec } x.t/x]$$

$$(\lambda x.t) u \sim t[u/x]$$

$$\pi_\alpha(\alpha t) \sim t$$

$$\pi_\beta(\alpha t) \sim \emptyset \quad \text{if } \alpha \neq \beta$$

$$[!u > !x \Rightarrow t] \sim t[u/x]$$

$$[\sum_{i \in I} u_i > !x \Rightarrow t] \sim \sum_{i \in I} [u_i > !x \Rightarrow t]$$

Theorem Bisimilarity is a congruence for HOPLA
Proof. Using Howe's method.

Expressive power

- type of CCS processes: $\mathbb{P} = \sum_{\alpha} !\mathbb{P}$

$$\begin{array}{ll}
 \mathcal{H}[[x]] \equiv_{\text{def}} x & \mathcal{H}[[t|u]] \equiv_{\text{def}} \text{Par } \mathcal{H}[[t]] \ \mathcal{H}[[u]] \\
 \mathcal{H}[[\text{rec } x.t]] \equiv_{\text{def}} \text{rec } x.\mathcal{H}[[t]] & \mathcal{H}[[t \setminus S]] \equiv_{\text{def}} \text{Res}_S \ \mathcal{H}[[t]] \\
 \mathcal{H}[[\sum_{i \in I} t_i]] \equiv_{\text{def}} \sum_{i \in I} \mathcal{H}[[t_i]] & \mathcal{H}[[t[f]]] \equiv_{\text{def}} \text{Rel}_f \ \mathcal{H}[[t]] \\
 \mathcal{H}[[\alpha.t]] \equiv_{\text{def}} \alpha!\mathcal{H}[[t]] &
 \end{array}$$

- parallel composition in CCS

$$\frac{t \xrightarrow{\alpha} t'}{t|u \xrightarrow{\alpha} t'|u} \quad \frac{t \xrightarrow{l} t' \quad u \xrightarrow{\bar{l}} u'}{t|u \xrightarrow{\tau} t'|u'} \quad \frac{u \xrightarrow{\alpha} u'}{t|u \xrightarrow{\alpha} t|u'}$$

translated into $\text{Par} : \mathbb{P} \rightarrow (\mathbb{P} \rightarrow \mathbb{P})$

$$\begin{aligned}
 \text{Par} \equiv_{\text{def}} & \text{rec } f.\lambda x.\lambda y.\sum_{\alpha} [\pi_{\alpha} x > !x' \Rightarrow \alpha!(f \ x' \ y)] + \\
 & \sum_{\alpha} [\pi_{\alpha} y > !y' \Rightarrow \alpha!(f \ x \ y')] + \\
 & \sum_l [\pi_l x > !x' \Rightarrow [\pi_{\bar{l}} y > !y' \Rightarrow \tau!(f \ x' \ y')]]
 \end{aligned}$$

Outline

- a taste of concurrency theory
 - models
 - languages
 - equivalences
- a simple domain theory for concurrency
 - path semantics
 - HOPLA
 - relating semantics
- extensions
 - strong correspondence
 - Affine HOPLA
 - related and future work

Types and paths

- types are built inductively from *paths*

$$p, q ::= P \mapsto q \mid \alpha p \mid P \mid abs\ p$$

- type of CCS processes: $\mathbb{P} = \Sigma_{\alpha} !\mathbb{P}$

$$\frac{p_1 : \mathbb{P} \ \cdots \ p_k : \mathbb{P}}{abs\ \alpha\{p_1, \dots, p_k\} : \mathbb{P}}$$

I **a**cept money
then give either
beer or **c**offee



$$a!(b!\emptyset + c!\emptyset)$$

$$\{a\emptyset, a\{b\emptyset\}, a\{c\emptyset\}, a\{b\emptyset, c\emptyset\}\}$$

I either
acept money
then give **b**eer
or
acept money
then give **c**offee



$$a!b!\emptyset + a!c!\emptyset$$

$$\{a\emptyset, a\{b\emptyset\}, a\{c\emptyset\}\}$$

Soundness and adequacy

Proposition (soundness) $t \xrightarrow{!} t' \implies \llbracket !t' \rrbracket \subseteq \llbracket t \rrbracket$

Proof. By rule induction on the operational rules.

Proposition (adequacy) $\llbracket t \rrbracket \neq \emptyset \iff \exists t'. t \xrightarrow{!} t'$

Proof. Using logical relations.

$$X \subseteq_{\mathbb{P}} t \iff_{\text{def}} \forall p \in X. p \in_{\mathbb{P}} t$$

$$P \mapsto q \in_{\mathbb{P} \rightarrow \mathbb{Q}} t \iff_{\text{def}} \forall u. (P \subseteq_{\mathbb{P}} u \implies q \in_{\mathbb{Q}} t u)$$

$$\alpha p \in_{\Sigma_{\alpha \in A} \mathbb{P}_{\alpha}} t \iff_{\text{def}} p \in_{\mathbb{P}_{\alpha}} \pi_{\alpha} t$$

$$P \in_{! \mathbb{P}} t \iff_{\text{def}} \exists t'. t \xrightarrow{!} t' \text{ and } P \subseteq_{\mathbb{P}} t'$$

$$\text{abs } p \in_{\mu_j \vec{T}. \vec{T}} t \iff_{\text{def}} p \in_{\mathbb{T}_j[\mu \vec{T}. \vec{T} / \vec{T}]} \text{rep } t$$

Lemma $\llbracket t \rrbracket \subseteq_{\mathbb{P}} t$

Proof. By structural induction on t .

Full abstraction

- program: a closed term of type $!⊙$
- observations for programs: $t \xrightarrow{!}$
- program context for t : C such that $C(t)$ a program
- *contextual preorder*: $t_1 \sqsubseteq t_2$ if

$$\forall C. C(t_1) \xrightarrow{!} \implies C(t_2) \xrightarrow{!}$$

Theorem (full abstraction) $\llbracket t_1 \rrbracket \subseteq \llbracket t_2 \rrbracket \iff t_1 \sqsubseteq t_2$

Proof. Define suitable program contexts $C_p(-)$:

$$\llbracket C_p(t) \rrbracket \neq \emptyset \iff p \in \llbracket t \rrbracket$$

Proof of full abstraction

- path “consumers”:

(cf. McCusker’02)

$$\begin{aligned}C_{P \mapsto q} &\equiv_{\text{def}} C_q(- t'_P) \\ C_{\beta p} &\equiv_{\text{def}} C_p(\pi_\beta -) \\ C_P &\equiv_{\text{def}} [- > !x \Rightarrow C'_P(x)] \\ C_{\text{abs } p} &\equiv_{\text{def}} C_p(\text{rep } -)\end{aligned}$$

- path “realisers”:

$$\begin{aligned}t_{P \mapsto q} &\equiv_{\text{def}} \lambda x. [C'_P(x) > !x' \Rightarrow t_q] \\ t_{\beta p} &\equiv_{\text{def}} \beta t_p \\ t_P &\equiv_{\text{def}} !t'_P \\ t_{\text{abs } p} &\equiv_{\text{def}} \text{abs } t_p\end{aligned}$$

- consumers and realisers of finite sets of paths:

$$\begin{aligned}C'_{\{p_1, \dots, p_n\}} &\equiv_{\text{def}} [C_{p_1} > !x_1 \Rightarrow \dots \Rightarrow [C_{p_n} > !x_n \Rightarrow !\emptyset] \dots] \\ t'_{\{p_1, \dots, p_n\}} &\equiv_{\text{def}} t_{p_1} + \dots + t_{p_n}\end{aligned}$$

A logical characterisation

- fragment of Hennessy-Milner logic $\phi ::= \langle a \rangle \phi \mid \bigwedge_{i \leq n} \phi_i$
- characteristic for *simulation equivalence* (image-finite)
- *logical preorder*: $t_1 \sqsubseteq_L t_2$ if $\forall \phi. t_1 \models \phi \implies t_2 \models \phi$

Theorem $t_1 \sqsubseteq t_2 \iff t_1 \sqsubseteq_L t_2$

Proof. Use suitable program contexts $C_\phi(-)$:

$$C_\phi(t) \xrightarrow{!} \iff t \models \phi$$

A simple domain theory for concurrency

- path sets form nondeterministic domains
- HOPLA obtained from canonical constructions on path sets
- simple operational semantics
- standard bisimulation congruence
- good expressive power
- simple proofs of soundness and adequacy
- simple proof of full abstraction
- path equivalence = contextual equivalence = logical equivalence = simulation equivalence

Thesis

Concurrent computation can be given
an abstract mathematical treatment
similar to that provided for sequential computation
by domain theory and denotational semantics
of Scott and Strachey

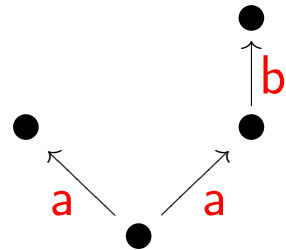
Outline

- a taste of concurrency theory
 - models
 - languages
 - equivalences
- a simple domain theory for concurrency
 - path semantics
 - HOPLA
 - relating semantics
- extensions
 - strong correspondence
 - Affine HOPLA
 - related and future work

Strong correspondence

- path semantics does not capture all branching information

I either
 accept money
 or
 accept money
 then give beer

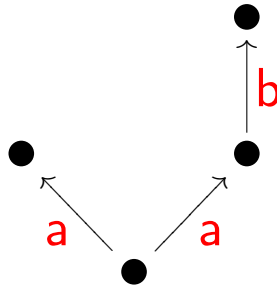


$$\llbracket a!\emptyset + a!b!\emptyset \rrbracket = \llbracket a!b!\emptyset \rrbracket$$

- more informative domain theory based on presheaves

$a\{b\emptyset\}$ ✓

$a\emptyset$ ✓



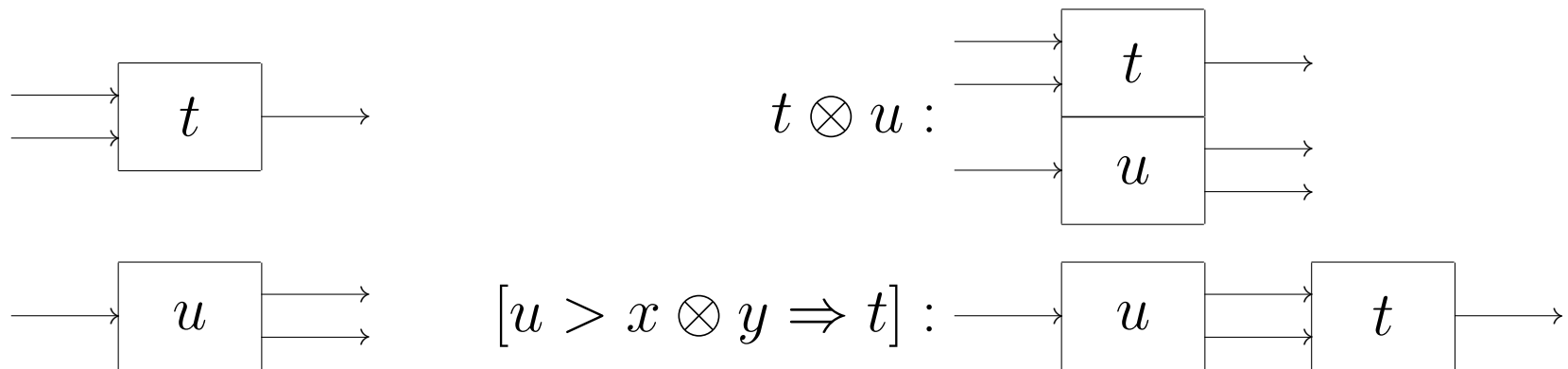
$a\{b\emptyset\}$

$a\emptyset$

Conjecture $\llbracket t \rrbracket \cong \sum_d \llbracket !t_d \rrbracket$ where $d : t \xrightarrow{!} t_d$.
 (Derivations correspond to elements of presheaves)

Affine HOPLA

- replace ! by $(-)_\perp$ to get model of *affine-linear logic*
- Affine HOPLA $t, u ::= \dots \mid t \otimes u \mid [u > x \otimes y \Rightarrow t]$



- operational semantics for first-order fragment
- soundness and adequacy, strong correspondence
- path semantics is fully abstract
- event-structure representation \rightsquigarrow “stable” semantics

Related and future work

- name-generation (Winskel&Zappa Nardelli'03)
- independence
 - Affine HOPLA can't do feed-back loops in dataflow
 - Affine HOPLA can't do “true concurrency” CCS
 - possible in underlying presheaf semantics
- bisimulation
 - presheaf semantics has built-in notion of bisimulation
 - automatically a congruence (Cattani&Winskel'96)
 - lack operational characterisation

Publications

With Glynn Winskel:

- Domain theory for concurrency, 2003. To appear in TCS
- Full abstraction for HOPLA, CONCUR'03
- HOPLA—a higher-order process language, CONCUR'02
- Linearity in process languages, LICS'02