Concurrency and Process Logics

Ken Stevens

Language Review

$\mathcal{E}$ ::=	A	constant
$\mid$	$\alpha.E$	prefixing
$\mid$	$\sum_{i\in I} E_i$	summation
$\mid$	$E_1 \mid E_2$	composition
$\mid$	$E[f]$	relabeling
$\mid$	$E\backslash\mathcal{L}$	restriction

$$\text{LTS} = (S, T, \{\xrightarrow{t} : t \in T\})$$

In composition, a label and colabel interact to form a single indivisible communication action τ.

Transition Semantics

SORTS:

Definition: for any $L \subseteq \mathcal{L}$, if the actions of P and all its derivatives lie in $L \cup \{\tau\}$ then we say P *has sort* L, or L *is a sort of* P, and write $P : L$.

Proposition: For every E and L, L is a sort of E if and only if, whenever $E \xrightarrow{\alpha} E'$, then

1. $\alpha \in L \cup \{\tau\}$
2. L is a sort of E'

Is $l \in E$??

Transition Semantics

SORTS:

Definition: for any $L \subseteq \mathcal{L}$, if the actions of P and all its derivatives lie in $L \cup \{\tau\}$ then we say P *has sort* L, or L *is a sort of* P, and write $P : L$.

Proposition: For every E and L, L is a sort of E if and only if, whenever $E \xrightarrow{\alpha} E'$, then

1. $\alpha \in L \cup \{\tau\}$
2. L is a sort of E'

Is $l \in E$?? Undecidable!!

Syntactic Sort

Given constants $\mathcal{L}(A)$ and variables $\mathcal{L}(x)$, syntactic sort $\mathcal{L}(E)$ of each agent expression E is defined as:

$$
\begin{array}{lcl}
\mathcal{L}(l.E) & = & \{l\} \cup \mathcal{L}(E) \\
\mathcal{L}(\tau.E) & = & \mathcal{L}(E) \\
\mathcal{L}(\sum_{i\in I} E_i) & = & \cup_{i\in I} \mathcal{L}(E_i) \\
\mathcal{L}(E \mid F) & = & \mathcal{L}(E) \cup \mathcal{L}(F) \\
\mathcal{L}(E \backslash L) & = & \mathcal{L}(E) - (L \cup \overline{L}) \\
\mathcal{L}(E[f]) & = & \{f(l) : l \in \mathcal{L}(E)\} \\
\text{if } A \stackrel{\text{def}}{=} P & = & \mathcal{L}(P) \subseteq \mathcal{L}(A)
\end{array}
$$

Syntactic Sort

Proposition Let $E \xrightarrow{\alpha} E'$ then

1. $\alpha \in \mathcal{L}(E) \cup \{\tau\}$
2. $\mathcal{L}(E') \subseteq \mathcal{L}(E)$

Proof by transition induction

Example:

```
Foo = a.b.c.Nil ;
sort Foo = { a, b, c }
sort Foo \{b} = { a, c }  <-- syntactic
min sort Foo \{b} = { a }
```

We will use min sort from here on out

Inference Proofs

$$((a.E + b.Nil) \mid \overline{a}.F)\backslash\{a\} \xrightarrow{\tau} (E \mid F)\backslash\{a\}$$

```
    Act   ------------
          a.E -a-> E
                |
                |
Sum1 --------------------        Act   --------------
      a.E + b.Nil -a-> E                'a.F -'a-> F

                \                          /
                 \                        /
       Com3  ----------------------------------
              (a.E + b.Nil) | 'a.F -tau-> E|F
                                |
                                |
 Res    ---------------------------------------------
         ((a.E + b.Nil) | 'a.F)\{a} -tau-> (E|F)\{a}
```

Inference Proofs

Infer the action: $(A \mid B)\backslash\{c\} \xrightarrow{a} (A' \mid B)\backslash\{c\}$

```
      Act  ---------------
            a.A' -a-> A'
                 |
                 |
      Con  ---------------   (A =def=> a.A')
             A -a-> A'
                 |
                 |
      Com1 ---------------
            A|B -a-> A'|B
                 |
                 |
Res   ---------------------------
      (A|B)\{c} -a-> (A'|B)\{c}
```

SECTION 4

Classification of Combinators

Two classifications:

1. Static
    - combinator remains after application – persistence
    - only part that has changed are those that have derivative actions
    - “Operators on Flow Graphs”
2. Dynamic
    - Combinator disappears after application – not persistent

The **Expansion Law**

- relates one group to another
- gives actions of static combinators in terms of themselves

Classification of Combinators

Static	Composition
	Restriction
	Relabeling
Dynamic	Act
	Summation
	Constants

Dynamic Laws

This allows us to **axiomatize** the language through **equational theory**

Monoid Laws

(1)	$P + Q = Q + P$	(symmetric)
(2)	$P + (Q + R) = (P + Q) + R$	(associative)
(3)	$P + P = P$	
(4)	$P + Nil = P$	

When we write '$=$' in the laws, we mean they have same derivatives:

$E_1 = E_2$ $\qquad$ $E_1 \xrightarrow{\alpha} E'$ iff $E_2 \xrightarrow{\alpha} E'$

Dynamic Laws

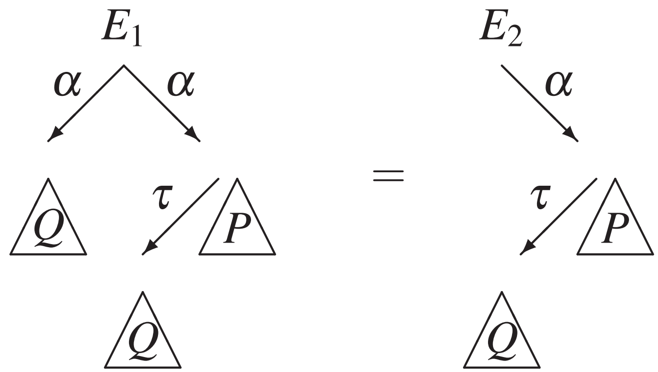
τ Laws

(1) $\alpha.\tau.P = \alpha.P$

(2) $P + \tau.P = \tau.P$

(3) $\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$

τ law (3) derivation trees (non-determinism of labels vs τ):



Note: α-derivatives of 2 agents differ!

$E_1 \xrightarrow{\alpha} E'$ $\qquad E_2 \stackrel{\alpha}{\not\rightarrow} E'$ $\qquad$ PROBLEM!!!

Dynamic Laws

Need a relation that supports $E_1 \xrightarrow{\alpha} Q'$ and $E_1 \xrightarrow{\alpha}\xrightarrow{\tau} Q$ as a native transition:

$P \overset{\alpha}{\Rightarrow} P'$ if $P(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^* P'$

Note that $(\xrightarrow{\tau})^*$ is the transitive closure of τ actions (0 or more $\xrightarrow{\tau}$)

Prove with τ laws 2 and 3, so:

$E_1 \overset{\alpha}{\Rightarrow} E'$ iff $E_2 \overset{\alpha}{\Rightarrow} E'$

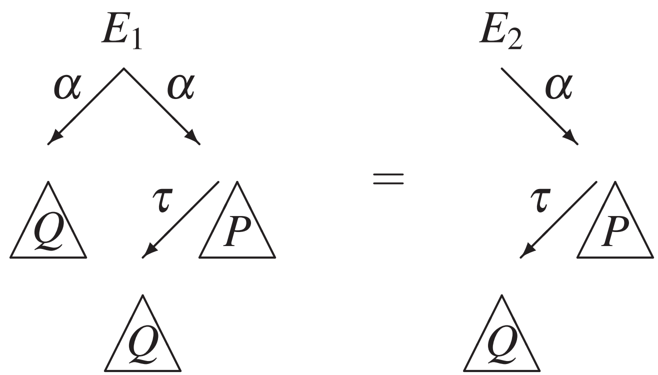
So $\overset{\alpha}{\Rightarrow}$ derivatives are the same!!

Dynamic Laws

Now we can define an equivalence relation that holds given τ transitions!

(3) $\alpha.(P+\tau.Q)+\alpha.Q=\alpha.(P+\tau.Q)$

τ law (3) derivation trees (non-determinism of labels vs τ):



Note: α-derivatives of 2 agents now the same!

$E_1 \overset{\alpha}{\Rightarrow} E'$ $\qquad$ $E_2 \overset{\alpha}{\Rightarrow} E'$

Example Proof

$$\alpha.(P + \tau.\tau.P) = \alpha.P$$

$\alpha.(P + \tau.P)$	τ(1)
$\alpha.\tau.P$	τ(2)
$\alpha.P$	τ(1)

Why reject some laws?

Could we prove:

$$\tau.P = P' \quad ??$$

Why reject some laws?

Could we prove:

if	$\tau.P = P'$
then	$a.P + \tau.b.Q = a.P + b.Q$

if	$\alpha.(P+Q) = \alpha.P + \alpha.Q$	(distributive)
then	$a.(b.P + c.Q) = a.b.P + a.c.Q$	

Why reject some laws?

Could we prove:

if	$\tau.P = P'$
then	$a.P + \tau.b.Q = a.P + b.Q$

if	$\alpha.(P+Q) = \alpha.P + \alpha.Q$	(distributive)
then	$a.(b.P + c.Q) = a.b.P + a.c.Q$	

These don't make sense!

$E_1 \stackrel{\alpha}{\Rightarrow} E' \qquad E_2 \stackrel{\alpha}{\not\Rightarrow} E'$

Note **where** the decision is made!

SECTION 5

Recursive Equations

1. $$\frac{\text{Assume } A \stackrel{\text{def}}{=} P \text{ where } A \text{ occurs in } P}{\text{Therefore } P \text{ is of form } E\{A/X\}}$$

2. by defining $A \stackrel{\text{def}}{=} E\{A/X\}$ where E is agent expression, A is constant, and X is a variable

3. intends A is a solution of equation $X = E$ (variable is definition of expression).

No time this term... Yay!!!

Expansion Law

Relates static and dynamic combinators – hierarchy and behavior.

Expansion Law derives actions of agents in standard concurrent form.

Standard concurrent form: $(P_1 \mid \ldots \mid P_n)\backslash L$

Example:
(Jobber | Jobber | Hammer | Mallet)$\backslash\{getm, putm, geth, puth\}$

Many times P_i's are purely sequential, i.e. prefix and summation only.

Hardware agents at the lowest level (e.g. NAND gate)

Expansion law will derive all derivative actions from current expression.

Expansion Law

Two forms of actions from transitional laws:

- α of a single component, and $\alpha \notin L \cup \overline{L}$

  $(P_1[f_1] \mid \ldots \mid P_i[f_i] \mid \ldots \mid P_n[f_n]) \backslash L \xrightarrow{\alpha}$
  $(P_1[f_1] \mid \ldots \mid P_i'[f_i] \mid \ldots \mid P_n[f_n]) \backslash L$

  Only change is in i^{th} component.

- τ action

  $P_i \xrightarrow{l_1}$ and $P_j \xrightarrow{l_2}$ $(1 \leq i < j \leq n)$
  where $f_i(l_1) = \overline{f_j(l_2)}$

  $(P_1[f_1] \mid \ldots \mid P_i[f_i] \mid \ldots \mid P_j[f_j] \mid \ldots \mid P_n[f_n]) \backslash L \xrightarrow{\tau}$
  $(P_1[f_1] \mid \ldots \mid P_i'[f_i] \mid \ldots \mid P_j'[f_j] \mid \ldots \mid P_n[f_n]) \backslash L$

  Exactly two components have changed.

Expansion Law

Formally:

let $P = (P_1[f_1] \mid \ldots \mid P_n[f_n])\backslash L$ with $n \geq 1$ then

$$P = \sum\{f_i(\alpha).(P_1[f_1] \mid \ldots \mid P_i'[f_i] \mid \ldots \mid P_n[f_n])\backslash L : \\ P_i \xrightarrow{\alpha} P_i',\ f_i(\alpha) \notin L \cup \overline{L}\} \\ + \sum\{\tau.(P_1[f_1] \mid \ldots \mid P_i'[f_i] \mid \ldots \mid P_j'[f_j] \mid \ldots \mid P_n[f_n])\backslash L : \\ P_i \xrightarrow{l_1} P_i',\ P_j \xrightarrow{l_2} P_j',\ f_i(l_1) = \overline{f_j(l_2)},\ i < j\}$$

Simplifying for clarity such that $P[f] = P$

let $P = (P_1 \mid \ldots \mid P_n)\backslash L$ with $n \geq 1$ then

$$P = \sum\{\alpha.(P_1 \mid \ldots \mid P_i' \mid \ldots \mid P_n)\backslash L : P_i \xrightarrow{\alpha} P_i',\ \alpha \notin L \cup \overline{L}\} \\ + \sum\{\tau.(P_1 \mid \ldots \mid P_i' \mid \ldots \mid P_j' \mid \ldots \mid P_n)\backslash L : \\ P_i \xrightarrow{l_1} P_i',\ P_j \xrightarrow{l_2} P_j',\ l_1 = \overline{l_2},\ i < j\}$$

Expansion Law

"artificial" example from Milner:

$$P_1 = a.P_1' + b.P_1''$$
$$P_2 = \overline{a}.P_2' + c.P_2''$$
$$P = (P_1 \mid P_2)\backslash a$$

So, $P = b.(P_1'' \mid P_2)\backslash a + c.(P_1 \mid P_2'')\backslash a + \tau.(P_1' \mid P_2')\backslash a$

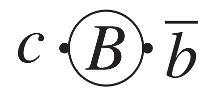
Further, assume

$$P_3 = \overline{a}.P_3' + \overline{c}.P_3''$$
$$Q = (P_1 \mid P_2 \mid P_3)\backslash\{a, b\}$$

(substituting L for $\{a, b\}$):

$$Q = c.(P_1 \mid P_2'' \mid P_3)\backslash L + \overline{c}.(P_1 \mid P_2 \mid P_3'')\backslash L$$
$$+\tau.(P_1' \mid P_2' \mid P_3)\backslash L + \tau.(P_1' \mid P_2 \mid P_3')\backslash L + \tau.(P_1 \mid P_2'' \mid P_3'')\backslash L$$

Expansion Law Example



$$A \stackrel{\text{def}}{=} a.A' \qquad B \stackrel{\text{def}}{=} c.B'$$

$$A' \stackrel{\text{def}}{=} \overline{c}.A \qquad B' \stackrel{\text{def}}{=} \overline{b}.B$$

Argued informally

$$(A \mid B)\backslash c = a.D \;\text{ where }\; D \stackrel{\text{def}}{=} a.\overline{b}.D + \overline{b}.a.D$$

Formally, apply expansion law:

$$(A \mid B)\backslash c = a.(A' \mid B)\backslash c$$
$$(A' \mid B)\backslash c = \tau.(A \mid B')\backslash c$$
$$(A \mid B')\backslash c = a.(A' \mid B')\backslash c + (A \mid B)\backslash c$$
$$(A' \mid B')\backslash c = \overline{b}.(A' \mid B)\backslash c$$

Applying $\alpha.\tau.P = \alpha.P$

$(A \mid B') = D$

so $(A \mid B) = a.(A \mid B')$

Expansion Law Example

By using Constant definitions, we can now turn hierarchcial description into a **canonical** form:

$$A \stackrel{\text{def}}{=} a.A' \qquad B \stackrel{\text{def}}{=} c.B'$$
$$A' \stackrel{\text{def}}{=} \overline{c}.A \qquad B' \stackrel{\text{def}}{=} \overline{b}.B$$

$$(A \mid B)\backslash c = E$$

where

$$E = a.E_1$$
$$E_1 = a.E_2 + \overline{b}.E$$
$$E_2 = \overline{b}.E_1$$

(E is the *minimized* form of $(A \mid B)$)

SECTION 6

Classification of Combinators

Static	Composition
	Restriction
	Relabeling
Dynamic	Act
	Summation
	Constants

The Static Laws

"Algebra of Flow Graphs"

- inner labels vs. outer labels
  - "library parts", connected with relabeling
- connected via l, $\bar{l}$

Static laws:

- $P \mid Q$ – joining every pair of ports with complementary labels
- $P \backslash L$ – erasing outer label l, $\bar{l}$ from $P . \forall l \in L$
- $P[f]$ – apply function f to all *outer* labels

The Static Laws

Composition Axiomitization

(1)	$P \mid Q = Q \mid P$	(symmetric)
(2)	$P \mid (Q \mid R) = (P \mid Q) \mid R$	(associative)
(3)	$P \mid Nil = P$	

The Static Laws

Restriction Axiomatization

(1) $P\backslash L = P$

if $\mathcal{L}(P) \cap (L \cup \overline{L}) = \emptyset$ (vacuous)

(2) $P\backslash K\backslash L = P\backslash(K \cup L)$

(3) $P[f]\backslash L = P\backslash f^{-1}(L)[f]$ (commutative*)

(4) $(P \mid Q)\backslash L = P\backslash L \mid Q\backslash L$

if $\mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \cap (L \cup \overline{L}) = \emptyset$ (distributive$^+$)

*: restriction and relabeling commute with some adjustment:

$f^{-1}(L) = \{l : f(l) \in L\}$

$^+$: restriction distributes over composition only if communications will not be restricted.

Static Laws

Examples

Assume FIFO is relabeled to use $mid1$ and $mid2$ for communication.

Then

(2): $(\text{FIFO} \mid \text{FIFO} \mid \text{FIFO})\backslash\{mid1\}\backslash\{mid2\} = \backslash\{mid1, mid2\}$

(4): $(\text{FIFO} \mid \text{FIFO})\backslash\{mid1\} \neq \text{FIFO}\backslash\{mid1\} \mid \text{FIFO}\backslash\{mid1\}$

The Static Laws

Relabeling Axiomatization

(1) $P[Id] = P$ (identity fn)

(2) $P[f] = P[f']$

if $f \upharpoonright \mathcal{L}(P) = f' \upharpoonright \mathcal{L}(P)$

(3) $P[f][f'] = P[f' \circ f]$

(4) $(P \mid Q)[f] = P[f] \mid Q[f]$

if $f \upharpoonright (L \cup \overline{L})$ is one-to-one

and where $L = \mathcal{L}(P \mid Q)$

Symbol $\upharpoonright$ restricts function to domain $\mathcal{L}(P)$

Symbol $\circ$ represents function composition: $f'(f(x))$

(4) is true if this will not create extra complementary port pairs.

f is one-to-one implies iff $x \neq y$ implies $f(x) \neq f(y)$

The Static Laws

Examples:

```
agent FIFO = a.'b.FIFO ;

(2): FIFO[mid1/b] = FIFO[mid1/b, mid2/g]

(3): FIFO[g/b][mid1/g] != FIFO[g/b, mid1/g]
```

Usually $[l'_i/l_i, \ldots, l'_n/l_n], l' \vee l$ distinct, $l'_i, \overline{l'_i} \notin \mathcal{L}(P)$

in this case, prop(4) usually applicable.

also for this case

$$[l'_i/l_i, \ldots, l'_n/l_n] = [l'_n/l_n] \circ \ldots \circ [l'_i/l_i]$$

so

$P[l'_i/l_i, \ldots, l'_n/l_n] = P[l'_n/l_n] \ldots [l'_i/l_i]$ by prop(4)

SECTION 6

Linear Time / Branching Time

Process Theory

Processes: The behavior of a system, machine, particle, protocol, etc.

E.g.: network of falling dominoes, chess players, etc.

Two activitites:

Modeling: Representing processes as elements of a mathematical domain $\ll$properties$\gg$ or expressions in a system description language llbehavioralgg.

Verification: Proving statements about processes

E.g.: whether two processes behave similarly, whether they have certain properties, (liveness, deadlock, etc.)

The verification constitutes the <u>semantics</u> of the laguage!

Comparative Concurrency Semantics

Process semantics are partially order by the relation:

“makes strictly more identifications on processes than”

truly creating a lattice of language strengths.

Comparative Concurrency Semantics

Semantic Notions of
Contemporary Process Theory

- **Linear Time** vs **Branching Time**

  “trace runs” “internal branching structure”

  To what extent should branching structure of execution path effect equality?

- Interleaving semantics vs Partial Orders

  To what extent should one identify processes differing in causal dependencies (while agreeing on possible orders of execution)?

- Abstractions to internal actions

  To what extent should we differentiate between processes differing only in internal or silent actions?

Comparative Concurrency Semantics

Semantic Notions of
Contemporary Process Theory

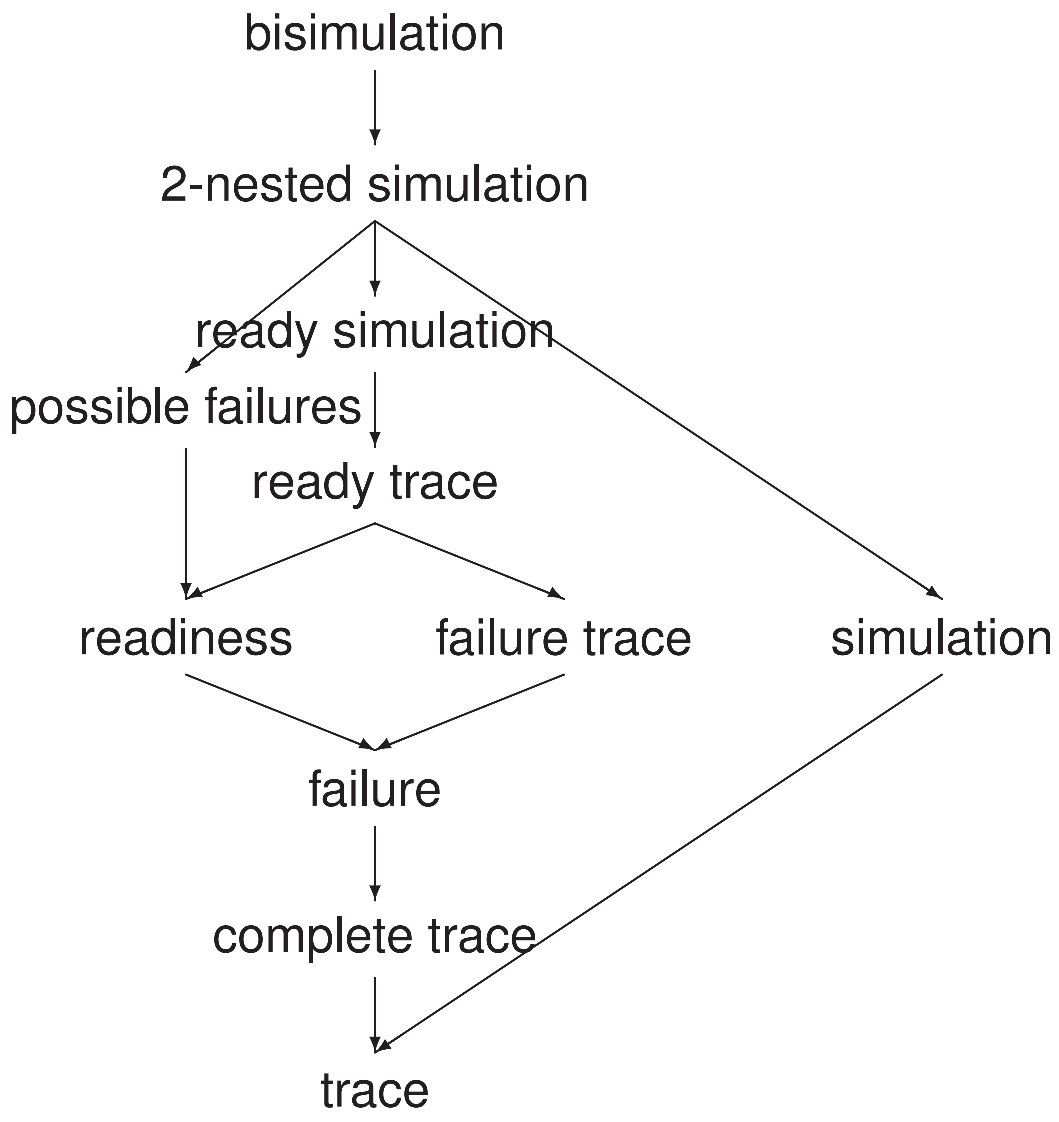
- Infinity
  What differences occur only in treating infinite behavior?
- Stochastic
- Real Time
- "Uniform Concurrency"
  Actions α, β, ... are not subject to further scrutiny.
  E.g.: Assignments to variables, moon launch, falling dominoes, signal voltage transition.

Comparative Concurrency Semantics

Limit to simple subset of above:

- Uniform concurrency
  - actions not subject to further scrutiny
- Sequential processes
  - Processes can perform one action at a time
- Finite Branching
  - from all states
- External observation
  - drop internal actions: CSP
  - "concrete" processes without internal actions: vanGlabeek
  - Modeled internal actions: CCS

Linear / Branching Time Spectrum



Linear / Branching Time Spectrum

- bisimulation
  - **CCS:** (park), observational equivalence (Hennesey & Milner, strong bisimulation all coincide on LTBT spectrum.
- 2-nested simulation
  - (Groote & Vaancrager)
- ready simulation
  - (bloom, Istrail, Meyer) “GSOS Trace Congruence” (Larsen/Skou) “2/3 bisimulation equivalence”
- ready trace
  - (pnuelli) called “barbed semantics”, also (Baeten Bergstom Klop) as “exhibited behavior semantics”

Linear / Branching Time Spectrum

- readiness
  - (Olerog, Hoar) slightly finer than failures
- failure trace
  - (philips) refusal semantics, **must equiv** in CWB
- Simulation
  - (park) independent of 5 semantics to left of lattice
- failure
  - **CSP:** (Brooks, Hoare, Roscoe), **testing** equivalence (DeNicola/Hennesey) for LTBT systems

Linear / Branching Time Spectrum

- complete trace
  - **may equivalence** in CWB
- Trace
  - (Hoar) – partial traces okay

Equivalences

On-board example of Job Shop

Look at Four Equivalences

- (weak)(complete) Trace Equivalence $=_t$
  - simple
  - not generally useful in arbitrary processes since it equates agents with different deadlock properties.
- Strong Equivalence $\sim$
  - useful but too strong
  - makes too many distinctions between agents
- Observation Equivalence (Bisimulation) $\approx$
  - The preferred notion of equivalence between agents
  - ...except that is is *not* a congruence (for summation).
    - Thus it does not admit equational reasoning
- Observational Congruence $=$

Look at Four Equivalences

The relationship of these four equivalence relations:

$P_1 \sim P_2 \supset P_1 = P_2 \supset P_1 \approx P_2 \supset P_1 =_t P_2$

All implications are proper

Venn diagrams complete inclusion