Concurrency and Process Logics

Ken Stevens

SECTION 6

Linear Time / Branching Time Spectrum

Process Theory

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

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

Two activities:

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 semantics of the language!

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

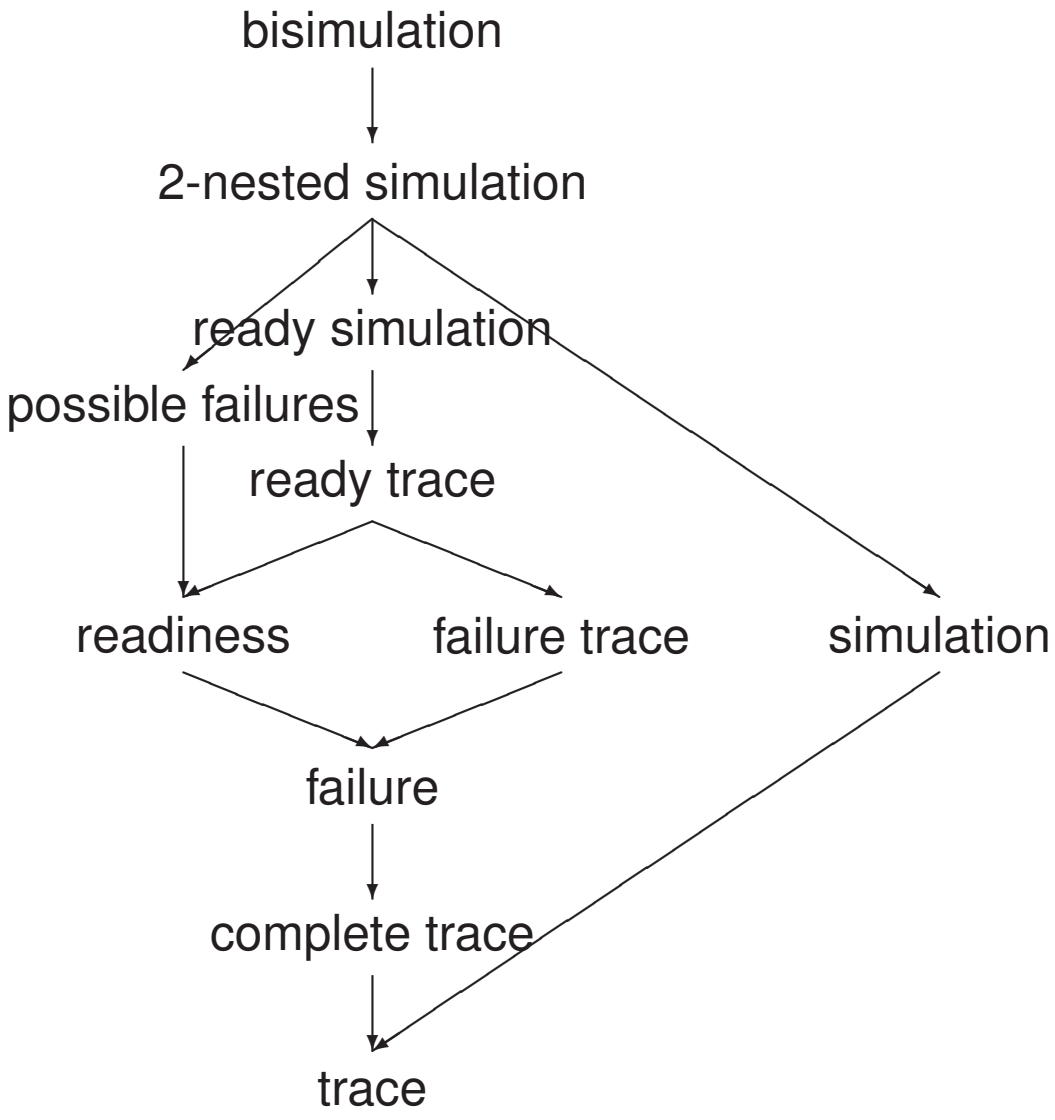
- Infinity
  What differences occur only in treating infinite behavior?
- Stochastic
- Real Time
- "Uniform Concurrency"
  Actions $\alpha, \beta, \ldots$ are not subject to further scrutiny.
  E.g.: Assignments to variables, moon launch, falling dominoes, signal voltage transition.

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

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

Equivalences

Interactive example of Job Shop

Section 7

Equivalences

(weak) (complete) Trace Semantics

1. LTS $= (S, T, \{\xrightarrow{\alpha} : \alpha \in T\})$

   S: set of states

   T: set of transition labels

   $\xrightarrow{\alpha} \subseteq S \times S$ transition relation $\forall \alpha \in T$

2. P is agent if $P \xrightarrow{\alpha} P'$ then α is action of P and P' is α-derivative of P

3. P is *derivation closed* if $\forall P \in \mathcal{P}$ and $\forall \alpha \in Act$ whenever $P \xrightarrow{\alpha} P'$ then $P' \in \mathcal{P}$

   When $\mathcal{P}$ is minimized (1 agent expression per state), size of $\mathcal{E}$ is state size

4. $s = \alpha_1 \ldots \alpha_n \in \mathcal{L}^*$ then $P \stackrel{s}{\Rightarrow} P'$ iff $P(\xrightarrow{\tau})^* \xrightarrow{\alpha_1} (\xrightarrow{\tau})^* \ldots (\xrightarrow{\tau})^* \xrightarrow{\alpha_n} (\xrightarrow{\tau})^* P'$
   (transitive, reflexive closure on τ)

   if $s = \varepsilon$ then $\stackrel{\varepsilon}{\Rightarrow} = (\xrightarrow{\tau})^*$

   Shorthand $P \stackrel{s}{\Rightarrow}$ stands for $P \stackrel{s}{\Rightarrow} P'$ for some P'

(weak) (complete) Trace Semantics

if $s \in Act^*, \hat{s}$ is the projection of s on $\mathcal{L}^*$

delete all τ in s

if $s \notin \mathcal{L}^*$ then $\hat{s} = \varepsilon$

if $s \in Act^*$ then $\hat{s} = \alpha_1, \ldots, \alpha_n \in \mathcal{L}^*$ and $P \stackrel{\hat{s}}{\Rightarrow} P'$ iff
$P(\stackrel{\tau}{\rightarrow})^* \stackrel{\alpha_1}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* \ldots (\stackrel{\tau}{\rightarrow})^* \stackrel{\alpha_n}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* P'$

(weak) (complete) Trace Semantics

Example

$\mathcal{E} = \{P_1, P_2\}, \quad P_1 \stackrel{def}{=} a.P_2; \;\; P_2 \stackrel{def}{=} \tau.P_1 + b.P_1;$

$P_1 \stackrel{a}{\rightarrow} P_1$ and $P_1 \stackrel{a}{\rightarrow} P_2$ are valid transitions

Trace Equivalence

s is a *trace* of P if $P \stackrel{s}{\Rightarrow} P'$ for some $P'(s \in \mathcal{L}^*)$

Trace equivalence is denoted as $P =_t Q$ iff $T(P) = T(Q)$

where $T(P) = \{s \in \mathcal{L}^* : P \stackrel{s}{\Rightarrow} P', \ \exists P'\}$

(weak) (complete) Trace Semantics

Formally,

P and Q are (weakly) (complete) trace equivalent, written $P =_t Q$ iff $\forall s \in \mathcal{L}^*$

$$P \stackrel{s}{\Rightarrow} \text{ iff } Q \stackrel{s}{\Rightarrow}$$

So $P =_t Q$ iff $\forall s \in \mathcal{L}^*$

if $P \stackrel{s}{\Rightarrow} P'$ for some P'

then $Q \stackrel{s}{\Rightarrow} Q'$ for some Q'

and if $Q \stackrel{s}{\Rightarrow} Q'$ for some Q'

then $P \stackrel{s}{\Rightarrow} P'$ for some P'

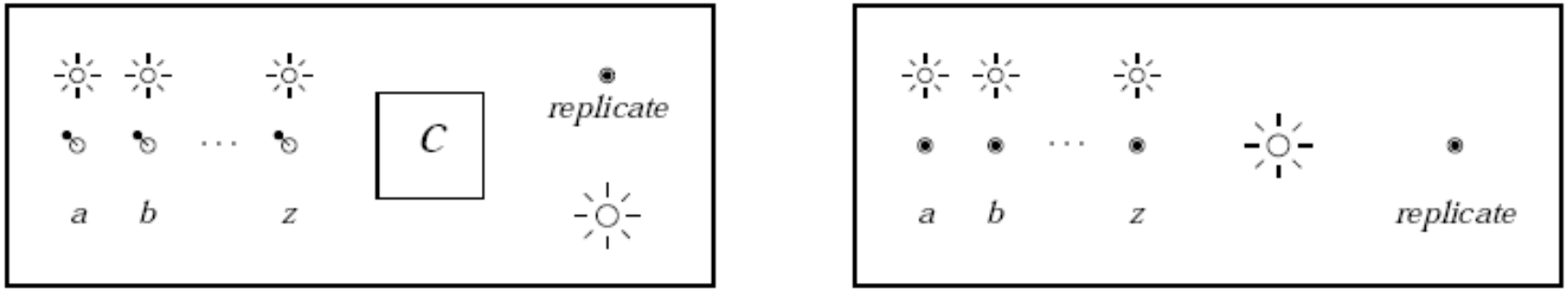
$T(P)$ *is the acceptance language for an automaton* – take a "formal languages and automaton course!

Testing Scenarios

Variety of testing scenarios available.

- Each scenario determines the notion of observable behavior
- The internal structure is not observable
- We will build more complex testing scenarios
  - generative
  - reactive

Generative and Reactive Testing



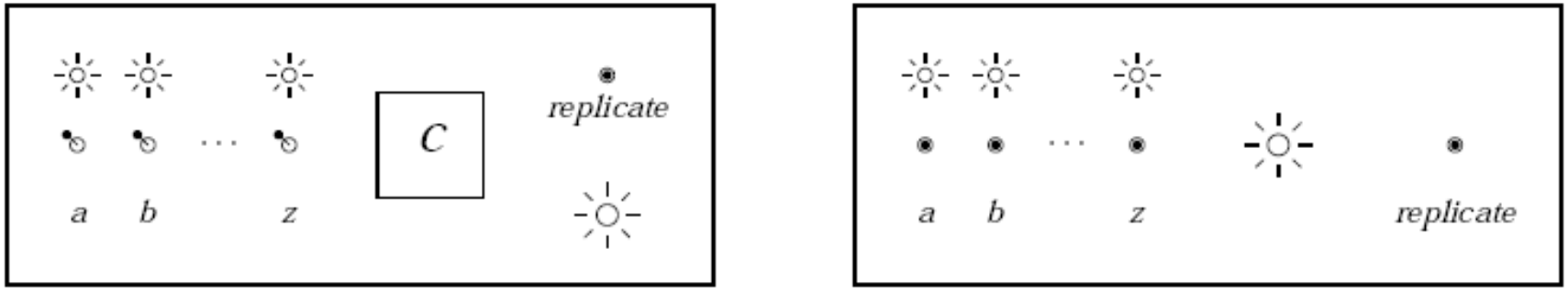
Generative and Reactive Machines

Generative machine spontaneously acts. Interaction is through *restricting* actions.

Reactive machine only acts in response to environment

Both are theoretically equivalent

Generative and Reactive Testing



Generative and Reactive Machines

- observable actions displayed when they occur. They stay until the display is clear.
- internal action light. (idle when no internal action)
- menu lights showing possible actions
- switches to block or create action
- replication button that creates concurrent copies from given state

Trace Semantics

$s \in Act^*$ is a *trace* of process P if $\exists P' : P \xrightarrow{s} P'$

Let $T(P)$ denotes the set of traces P.

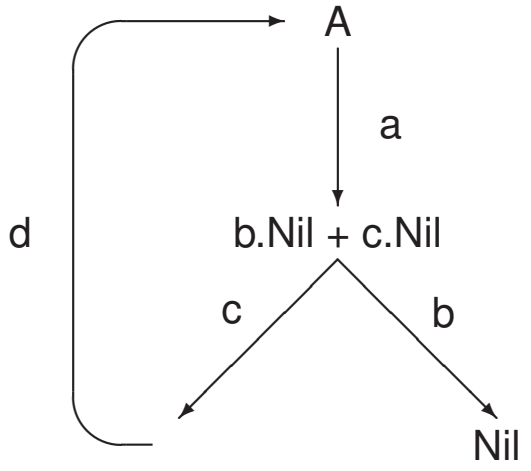
Two processes P and Q are trace equivalent, denoted $P =_t Q$ if $T(P) = T(Q)$

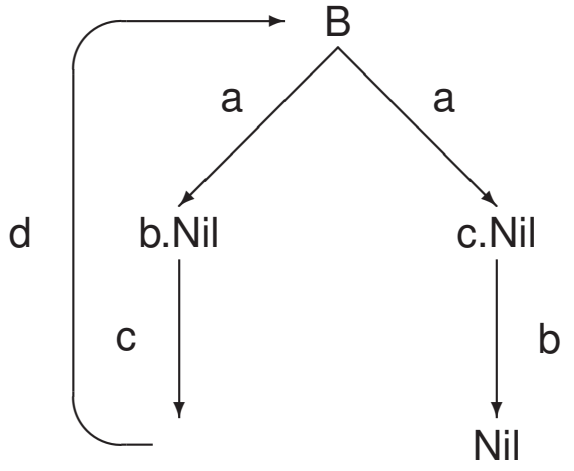
In trace semantics two processes are identified iff they are trace equivalent.

Trace Examples

$A \stackrel{def}{=} a.(b.Nil + c.d.A)$ $\qquad$ $B \stackrel{def}{=} a.b.Nil + a.c.d.A$

1. So A and B are finite state automata over alphabet $Act = \{a, b, c, d\}$
2. '$+$' is "union of languages", '.' is concatenation, Nil is the language who's only member is the empty string.
3. transition graphs:





Trace Example

Let's break agents down to look at states

$A \stackrel{def}{=} a.A_1$

$A_1 \stackrel{def}{=} b.A_2 + c.A_3$

$A_2 \stackrel{def}{=} Nil$

$A_3 \stackrel{def}{=} d.A$

$B \stackrel{def}{=} a.B_1 + a.B'_1$

$B_1 \stackrel{def}{=} b.B_2$

$B'_1 \stackrel{def}{=} c.B_3$

$B_2 \stackrel{def}{=} Nil$

$B_3 \stackrel{def}{=} d.B$

Now, let A_2 and B_2 be accepting states for the language.

(Automata would abbreviate CCS to remove Nil, but effect is the same.)

Trace Example

Using automata theory

$A \stackrel{def}{=} a.(b.Nil + c.d.A)$

Substitution to get 1st equation from concatenation and union

$A \stackrel{def}{=} a.b.Nil + a.c.d.A$

by the distributive law

$A \stackrel{def}{=} (a.c.d)^*.a.b$

deduce as regular grammar

$B \stackrel{def}{=} a.b.Nil + a.c.d.A$

Substitution to get 1st equation from concatenation and union

$A \stackrel{def}{=} (a.c.d)^*.a.b$

deduce as regular grammar

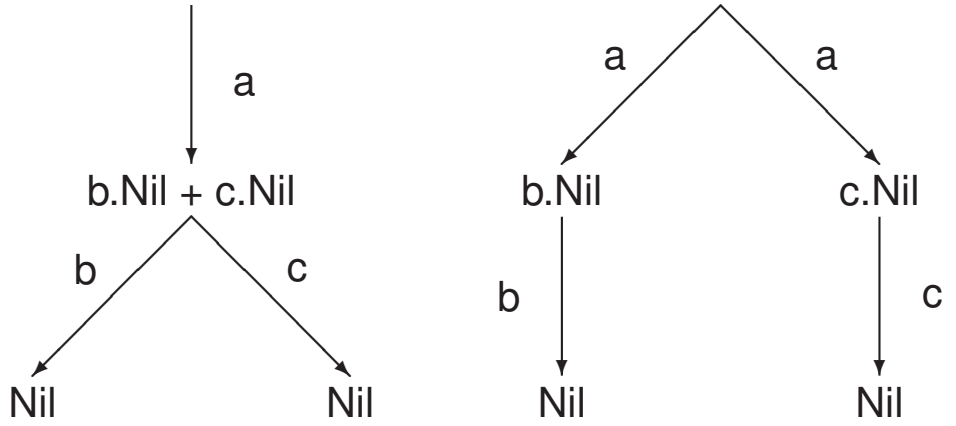
So these are equivalent using regular grammars and nondeterministic automata!

Trace Examples

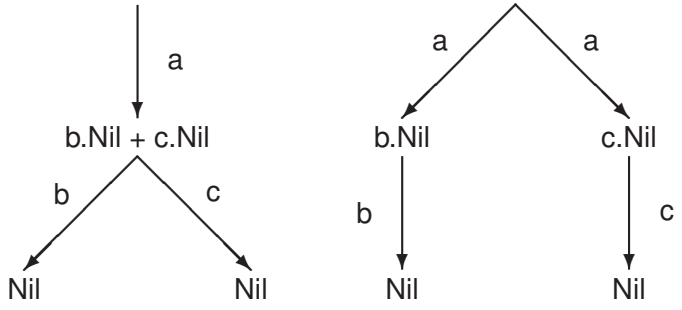
$$P \stackrel{def}{=} a.(b.Nil + c.Nil)$$

$$Q \stackrel{def}{=} a.b.Nil + a.c.Nil$$

Transition graphs:



Trace Examples

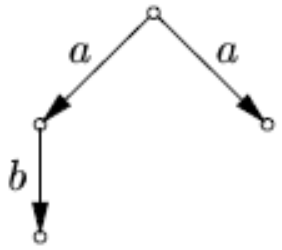


- $P =_t Q$
- But observable behavior different with respect to deadlock
- $(Q \mid R)$ can deadlock after action a
- reject this as general notion of *compositional* equivalence
- good for systems that reduce to a branch of *regular algebra* and *nondeterministic automata theory*

Complete Trace

$s \in Act^*$ is a *complete trace* of process P if $\exists P' : P \xrightarrow{s} P'$ and $I(P') = \emptyset$ (I = initial traces)

Two processes P and Q are trace equivalent, denoted $P =_{ct} Q$ iff $T(P) = T(Q)$ and $CT(P) = CT(Q)$



$ab + a$

$=_T$
$\neq_{CT}$
$=_S$
$\neq_F^1$

ab

$T \preceq CT$

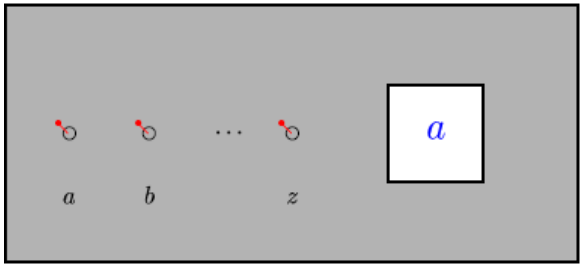
$T(left) = T(right) = \{\varepsilon, a, ab\}$

$a \in CT(left) - CT(right)$

Trace Testing

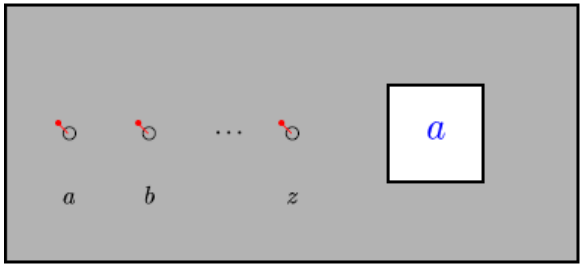
- Black box
- display shows name of action
- P autonomously chooses an execution path
- action always shown (except at start)
- observer records actions

Failures Semantics



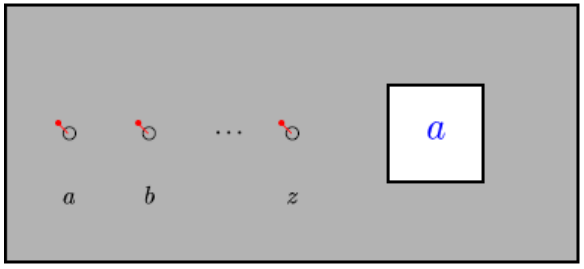
- Machine runs autonomously
- User chooses the signal sets that are enabled
- Machine will only permit the signals that are enabled
- Display turns off if deadlocked

Failures Semantics



- Same interface to outside world as trace semantics
- Adds switch for each action $a \in Act$
  - now can determine which actions are
    - free
    - blocked
- machine autonomously chooses execution path

Failures Semantics



Informally we can state this, based on our testing scenario, that:

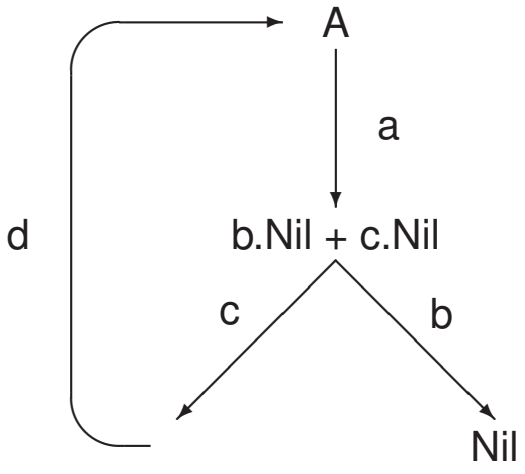
After a certain sequence of actions σ*, the set* X *of free actions is refused by the process.* X *is therefore called a* refusal set *and* $\langle \sigma, X \rangle$ *is a* failure pair*.*

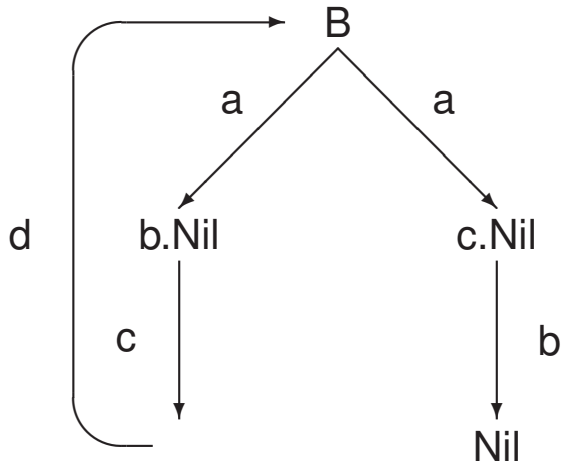
Failures Semantics

Failure is a pair (s, L) where $s \in \mathcal{L}^*$ is a trace and $L \subseteq \mathcal{L}$ is a set of labels. The failure (s, L) belongs to an agent P if $\exists P'$:

1. $P \stackrel{s}{\Rightarrow} P'$
2. $P' \stackrel{\tau}{\not\rightarrow}$ (so τ doesn't create $\alpha \in L$ actions)
3. $\forall \alpha \in L, P \stackrel{\alpha}{\not\rightarrow}$

Failure Semantics





$$A = (\varepsilon, \{b, c, d\})$$
$$A_1 = (\{a\}, \{a, d\})$$

$$B = (\varepsilon, \{b, c, d\})$$
$$B_1 = (\{a\}, \{a, c, d\})$$
$$B_1' = (\{a\}, \{a, b, d\})$$

If you initially block $\{b, c, d\}$ both machines respond the same. After that, different failure sets have different responses, showing they are not equivalent.

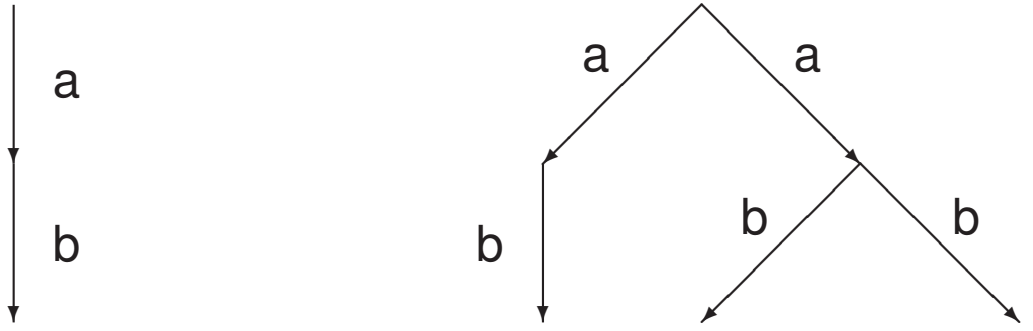
Failures Semantics

The distributive law of Automata Theory allowed the equivalence deduction:

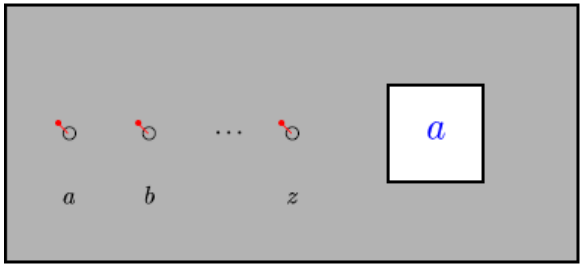
$$a.(P+Q) = a.P + a.Q$$

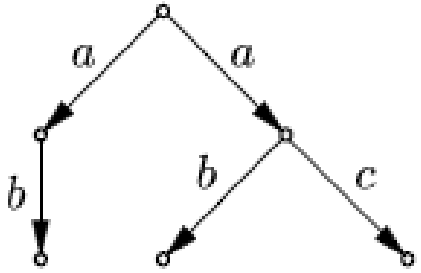
Failures semantics is not equivalent across distribution.

Note: Don't conclude that identical derivation trees required:



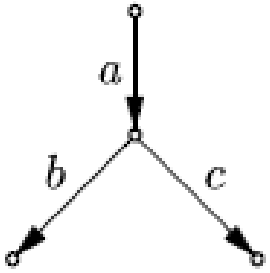
Trace and Failures Counterexample





$ab + a(b + c)$

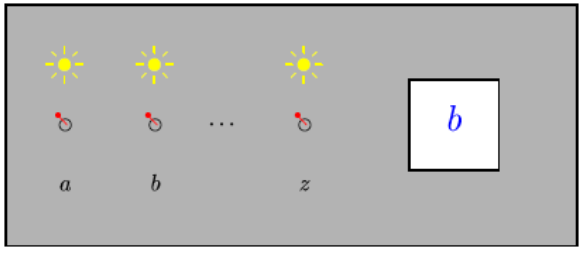
$=_{CT}$
$\neq_{F}$
$=_{CS}$
$\neq_{F}^{1}$



$a(b + c)$

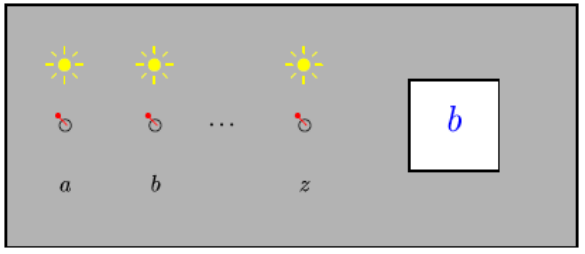
Ready Trace Semantics

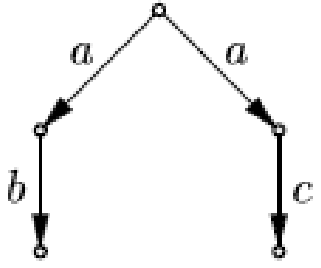
A variant of the failure trace machine equipped with a lamp for each action $a \in Act$. When process idles, lamps are lit if they can be engaged by the process. (Actions must be blocked by user to make process idle.)



This is important for **temporal logics** which can now directly see "observable" branching structure!

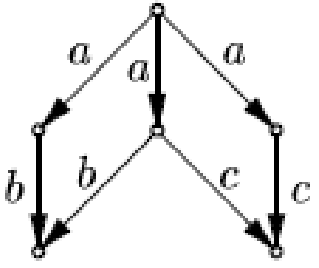
Ready Trace Counterexample





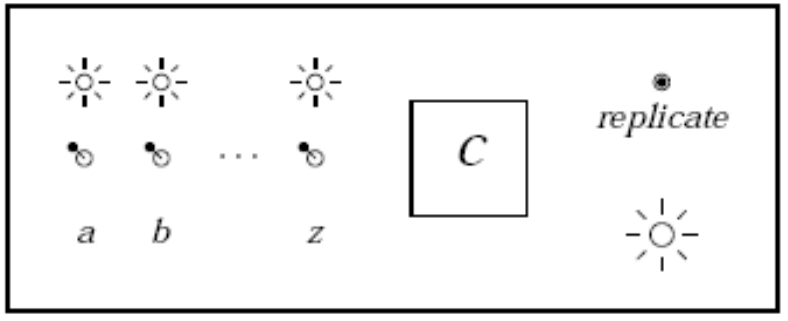
$ab + ac$

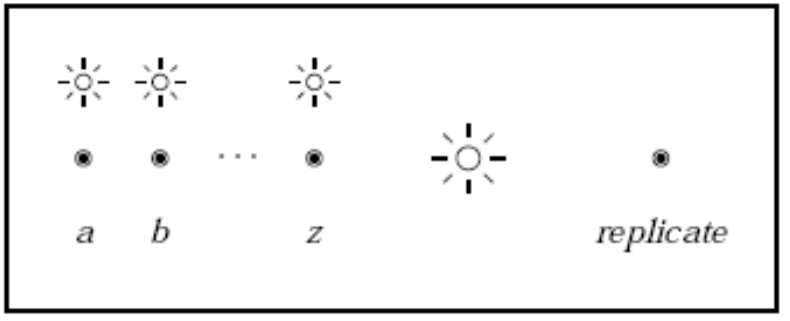
$=_F$
$\neq_R$
$=_{FT}$
$\neq_{RT}$



$ab + a(b + c) + ac$

Bisimulation Semantics





Reactive failure ability with duplicator and global testing capability.

"ability to enumerate all (of finitely many) possible 'operating environments' at each stage of the test, so as to guarantee that all nondeterministic branches will be pursued by various copies of the subject process"

Abramsky

Bisimulation Semantics

Global assumptions - example with environmental condition modalities on α:

1. It is the *weather* that determines which α move will occur from every state when α is pushed on the box.
2. The weather only has finitely many states for choice resolution.
3. We can control the weather.

So, if we investigate all α moves in all weather conditions, then we can describe the behavior of the system when carried out for all $\alpha \in \mathcal{L}(P)$

(finiteness can be relaxed, but important if you want a solution in a finite amount of time...)

Bisimulation Relation Extensions

Other extensions can be made to the relationship:

- Stochastic nature could be exploited here as the probabilities of the weather conditions could be listed.

Bisimulation Semantics

$P \approx Q$ iff $\forall \alpha \in Act$

(i) whenever $P \xrightarrow{\alpha} P'$ then, for some Q'

$Q \stackrel{\hat{\alpha}}{\Rightarrow} Q'$ and $P' \approx Q'$

(ii) whenever $Q \xrightarrow{\alpha} Q'$ then, for some P'

$P \stackrel{\hat{\alpha}}{\Rightarrow} P'$ and $P' \approx Q'$

Note:

if $\xrightarrow{\alpha}$ and $\alpha \neq \tau$

$$P \stackrel{\hat{\alpha}}{\Rightarrow} P' = P(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^* P'$$

if $\xrightarrow{\alpha}$ and $\alpha = \tau$

$$P \stackrel{\hat{\alpha}}{\Rightarrow} P' = P(\xrightarrow{\tau})^* P'$$

Bisimulation Relations

Observational Equivalence
$P \approx Q$ iff $P \zeta Q$ for some bisimulation ζ

Basic Properties

1. $\approx$ is an equivalence relation
   a. Idp (reflexive)
   b. S_i^{-1} (symmetric)
   c. $S_1 S_2$ (transitive)
2. $\approx$ is itself a bisimulation
   a. $\bigcup_{i \in I} S_i$
3. it satisfies the “back and forth” property

Bisimulation Relations

Example: Let

$$C_0 = \overline{b}.C_1 + a.C_2 \qquad D = \overline{b}.D_1 + a.D_2$$
$$C_1 = a.C_3 \qquad D_1 = a.D$$
$$C_2 = \overline{b}.C_3 \qquad D_2 = \overline{b}.D$$
$$C_3 = \tau.C_0$$

Assume $C_0 \approx D$ are bisimilar

Prove by constructing bisimular sets:

$$\zeta = \{ \quad (C_0, C_3, D) \quad (C_1, D_1) \quad (C_2, D_2) \}$$

Note: We want largest sets, as the sets (C_0, D) and (C_3, D) are also valid.

Other Issues

- τ cycles are ignored!
  - Possibility of active idling not detected in bisimulation.
  - This is different from *livelock*

Milner states

> *"In a theory which equates 2 agents even when the proportion between their speeds may vary unboundedly, it is natural to allow this proportion to be infinite."*

Other Issues

In fact:

Every system containing τ-cycles is equivalent to one without.

Nil $\approx A$ $\qquad\qquad$ $A \stackrel{def}{=} \tau.A$

If bisimulation is strengthened to include this, the awkward consequence follows:

$P \mid A \not\approx P$ where A is as defined above

Seems wrong since P is no less divergent when composed with τ-cycle running concurrently with it.

Strong Bisimulation

- Represented in Miler by relationship $=$
- Not very useful as it distinguishes τ as "observable" action

$P = Q$ iff $\forall \alpha \in Act$
$P \xrightarrow{\alpha} P'$ implies $(Q \xrightarrow{\alpha} Q'$ for some $Q' = P')$
and $Q \xrightarrow{\alpha} Q'$ implies $(P \xrightarrow{\alpha} P'$ for some $P' = Q')$

Congruence

Note: Bisimulation is not a congruence.

This is the case when the initial agent is divergent.

(A process is divergent or not stable if it can do a τ action)

This is due to the semantics of summation that we don't have time to show.

Circuit Verification

Bisimulation semantics gives **observational equivalence**

- basically any externally observable difference is recognized
- *allows full protocol compatibilility regardless of implementation!*

However, *equivalence* is NOT the relationship that we want!

- Equivalence to a specification is too strong a relationship.
- We need flexibility in inputs and in outputs

What we really need is a **conformance** relationship:

The specification *conforms*, or is a valid replacement of the implementation.

Results in difference between **specification** and **implementation**

Conformation

In a conformance relationship:

- the implementation can have a richer input behavior than the specification
  - the implementation can *(and must)* accept inputs in states that the specification does not allow.
    - essential in hardware to avoid *computation interference*
      - the condition when an input is passed to a process in a state where it cannot accept the input
  - e.g.: four-input mux design is valid for three-input mux specification.

Logic Conformation in Analyze

This is the conformance relationship implemented in Analyze:

A binary relation $\mathcal{LC} \subseteq \mathcal{P} \times \mathcal{P}$ over agents is a **logic conformation** between implementation I and specification S if $(I,S) \in \mathcal{LC}$ then $\forall\, \alpha \in Act$ and $\forall\, \beta \in \overline{\mathcal{A}} \cup \{\tau\}$ (outputs and τ) and $\forall\, \gamma \in \mathcal{A}$ (inputs)

(i) Whenever $S \xrightarrow{\alpha} S'$ then

$\exists\, I'$ such that $I \stackrel{\widehat{\alpha}}{\Rightarrow} I'$ and $(I',S') \in \mathcal{LC}$

(ii) Whenever $I \xrightarrow{\beta} I'$ then

$\exists\, S'$ such that $S \stackrel{\widehat{\beta}}{\Rightarrow} S'$ and $(I',S') \in \mathcal{LC}$

(iii) Whenever $I \xrightarrow{\gamma} I'$ and $S \stackrel{\gamma}{\Rightarrow}$ then

$\exists\, S'$ such that $S \stackrel{\gamma}{\Rightarrow} S'$ and $(I',S') \in \mathcal{LC}$

Logic Conformation in Analyze

This results in a partial order, not equivalence, relationship between implementation and specification, $S \prec I$

Conformance Weakening on Outputs

Outputs also do not need to equivalently match:

- the implementation can have a less rich output behavior when the outputs are **confluent**
  - Confluence occurs when the outputs can occur *concurrently*.
    - This is a relationship on labeled transition systems that matches true concurrency in languages such as petri nets.

1. If the specification allows concurrent output burst, then the composed implementation must accept all outputs.
2. If the implementation only produces a subset of the possible output sequences, then the system will still behave correctly.

Formalized by Brower.

Not implemented in Analyze