EQUIVALENCE CONCEPTS

Equivalence Relations are a very important aspect of *algebraic* specification techniques.

In other techniques (such as EFSM-based) we don’t have such extensive theory of equivalence.

Although proving equivalence of real-life specs is impossible with today’s means, a theory of equivalence is important for:

- precise understanding of language semantics

- tool development: tools may use the fact that some expressions are equivalent: e.g. A[]B = B[]A

- correctness-preserving transformation: transforming a specification written to emphasize certain aspects into a specification written to emphasize others

Equivalence-preserving transformations and Software Design

Design can start with a very abstract specification, representing the requirements.

Then, using equivalence-preserving transformations, this specification can be gradually transformed into an an implementation-oriented specification.

Finally, this one can be transformed into code.

Maintenance may require to replace some components with others, while maintaining the same system behavior.

Other equivalence-preserving transformation can make it possible to obtain various types of test cases (functional, black-box, white-box, etc.) from the specifications.

Tools to help in this process are available.

There is a lot of research in this area...

When are two processes EQUIVALENT?

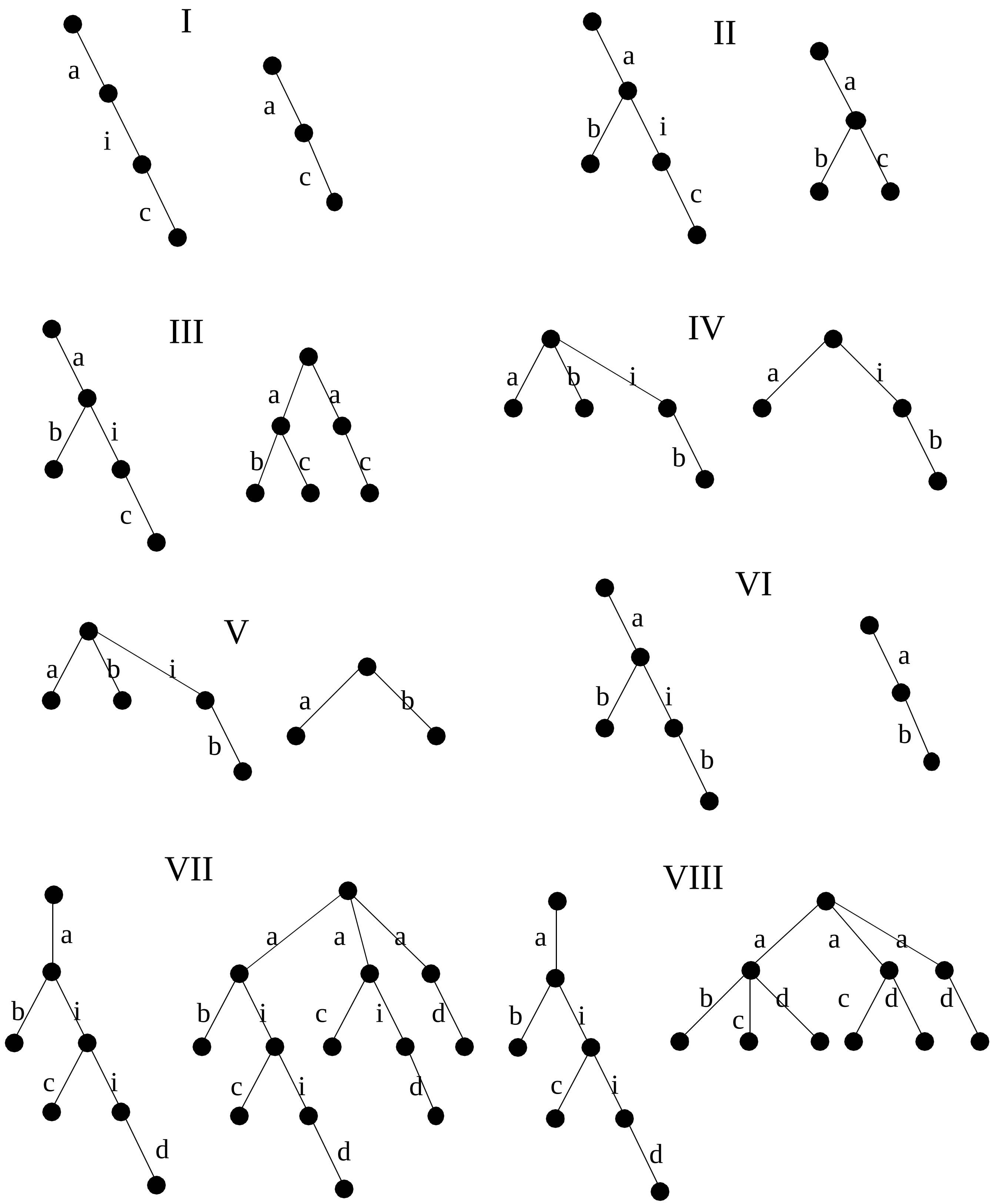
Some possibilities:

- whenever they can engage in identical action sequences (traces)
- whenever they have the same *observable behavior* (bisimulation)
- whenever no tester can distinguish them (testing equivalence)
- whenever they can be replaced one for the other in any system, yielding equivalent systems (congruences)
- whenever they have the same behavior trees (equality)

• • •

All these concepts will be examined in the course.

Others have been considered in the literature.



EQUIVALENT?

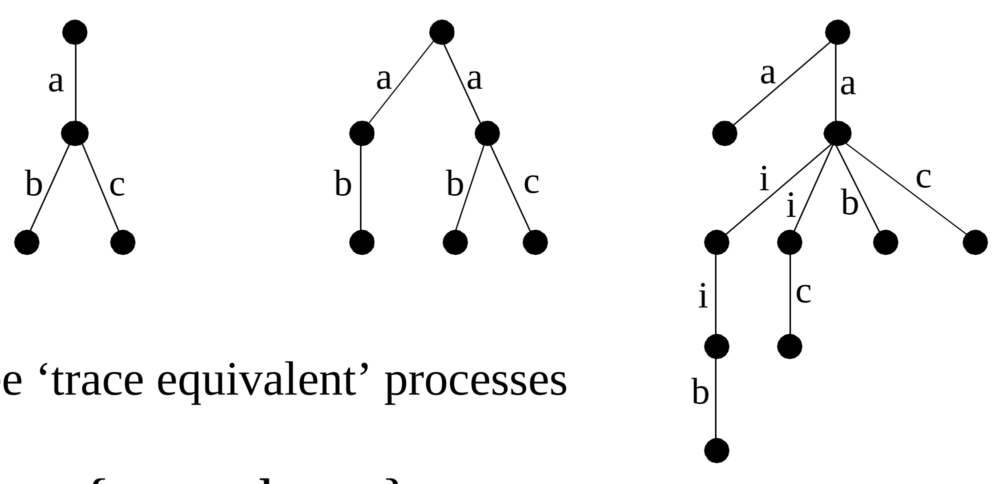
TRACE EQUIVALENCE

Perhaps the simplest possible equivalence concept:

> A *trace* is a sequence of observable actions
>
> Two processes are *trace equivalent* (***tr***) iff they can engage in the same traces

B=s=>, where s is a trace, means that B can engage in s, i.e. ∃B' s.th. B leads to B' after the observable sequence s, so

B1 *tr* B2 iff ∀s, B1=s=> iff B2=s=>



three 'trace equivalent' processes

Traces: {ε, a, ab, ac}

NB: the empty trace ε is always included (note that empty trace does not mean internal action)

IDENTITY = (ALSO ≡)

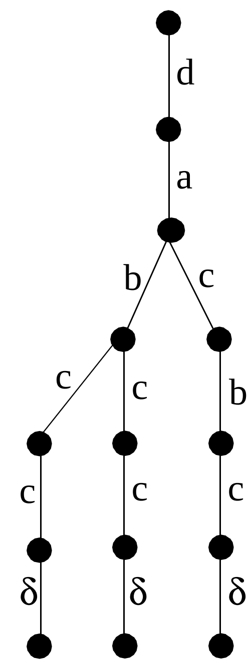
Behavior expressions have *identical beh. trees* (incl. all internal and external actions)

e.g.

a;b;c;**exit**
|[a]|
d;a;c;**exit**

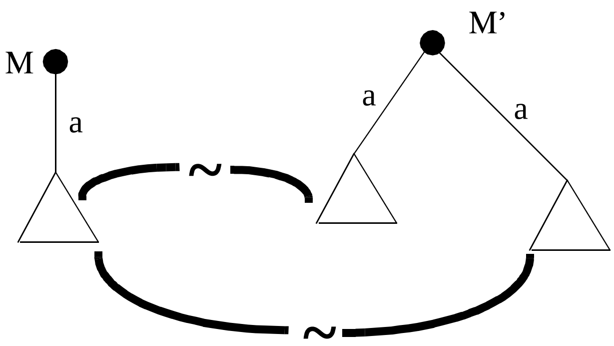
= d;a;(b;c;**exit** ||| c;**exit**)

= d;a;(b;(c;c;**exit** [] c;c;**exit**)[]c;b;c;**exit)**



The *expansion* laws are the best known identities.

To be discussed.



We can conclude

M ~ M’

BISIMULATION

Informally, it is reasonable to think that two b.exp. are *equivalent* if, whenever one of them can execute some action(s) leading to some behavior B, the other can execute the same action(s) leading to a behavior B’, where B is again *equivalent* to B’.

This is the basic idea of bisimulation: an inductive concept.

For two labelled transition systems S and S', we write

$$S \text{ -a -> } S' \qquad \text{(a: internal or external)}$$

if S can execute action a and become S'

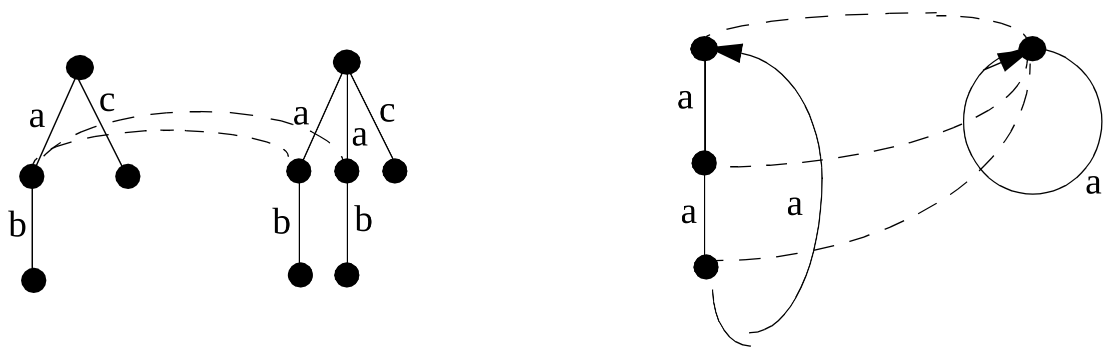
STRONG BISIMULATION $\sim$

A relation $\Re$ over LTS is a strong B. iff $\forall <s_1,s_2> \in \Re$, $\forall$ actions a (internal or external)

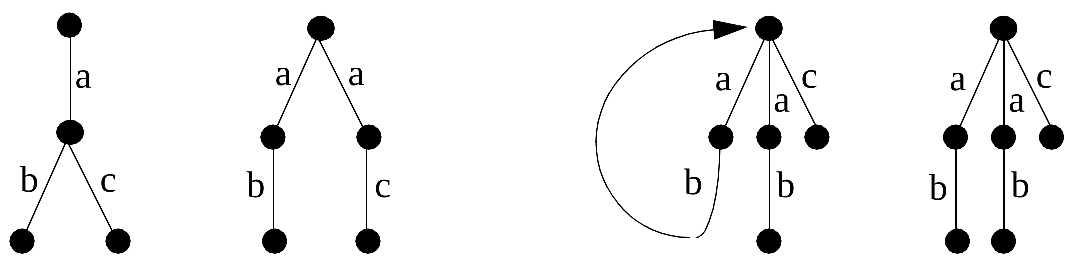
1) if $\exists\, s_1'$ s.th. s_1 - a -> s_1' then $\exists\, s_2'$ s.th. s_2 - a -> s_2' and $<s_1',s_2'> \in \Re$
2) and vice-versa (interchange s_1 and s_2)

Strongly Bisimilar LTS must be able to execute the same actions, and if they execute identical actions they must transform into strongly bisimilar LTS.

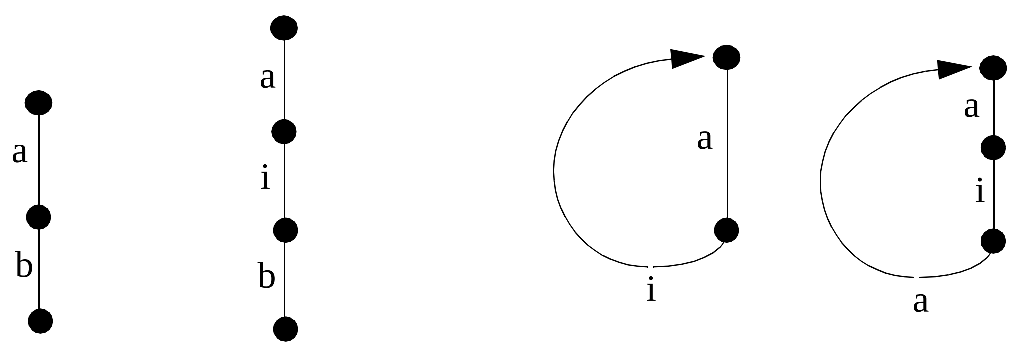
There is a strong bisimulation between:



But not between:

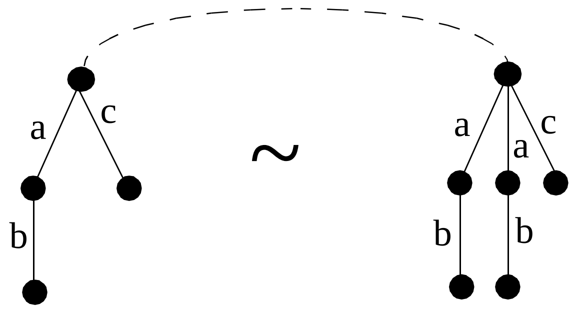


Nor between:



A ~ law which is not a = law:

B [] B ~ B



In this case, the relation ℜ is:

{<a;b;**stop** [] c;**stop**,   a;b;**stop** [] a;b;**stop** [] c;**stop**>} U Id

where Id is the set of all *identical* behavior pairs

In fact:

a;b;**stop** [] c;**stop** - a -> b;**stop**
a;b;**stop** [] a;b;**stop** [] c;**stop** -a -> b;**stop**

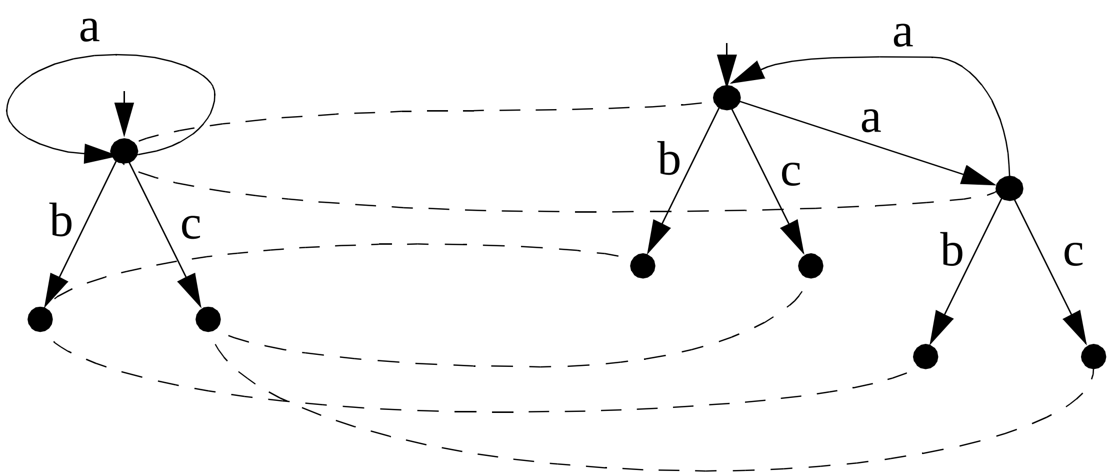
and <b;**stop**, b;**stop**> ∈ Id

Also

a;b;**stop** [] c;**stop** - c -> **stop**
a;b;**stop** [] a;b;**stop** [] c;**stop** -c -> **stop**

and <**stop**, **stop**> ∈ Id

This is symmetric because on the right there are no additional transitions.



A STRONG BISIMULATION

Strong bisimulation is fairly easy to compute, hence this relation is commonly used in tools to simplify LTSs.

ABSTRACTING FROM **i**

(to some extent)

i) Let s denote a string of actions, $s = a_1 \ldots a_n$

we write B-s->B' iff

$\exists\, B_1 \ldots B_n$ s.th. $B\text{-}a_1\text{->}B_1 \ldots B_{n-1}\text{-}a_n\text{->}B_n = B'$

we also write B-ε->B, where ε = empty string

ii) Let s denote a string of observable actions, $s = a_1 \ldots a_n$, and let i^k be a string of k internal actions

we write B=s=>B' whenever

$$\exists i^{k_0} a_1 i^{k_1} a_2 \ldots a_n i^{k_n}$$

(s with an arbitrary number of i interspersed) s.th.

$$B - i^{k_0} a_1 i^{k_1} a_2 \ldots a_n i^{k_n} \to B'$$

EXAMPLES:

Given a path on a LTS

$$B_0 \text{ - i -> } B_1 \text{ - a -> } B_2 \text{ - i -> } B_3 \text{ - b -> } B_4$$

we may write

B_0 - i a i b -> B_4
B_0 = a b => B_4
B_1 = a => B_2
B_0 = ε => B_1
B_0 = ε => B_0
B_0 = a => B_2
B_0 = a => B_3
etc.

Based on the observable sequence relation, we define a notion of *weak bisimulation*.

Definition

A relation $\Re$ between LTS is a *weak bisimulation* if for any pair <B1, B2> in $\Re$ and for any string *s* of observable actions:

whenever $B_1 = s \Rightarrow B_1'$, then for some B_2':

$B_2 = s \Rightarrow B_2'$ and $B_1' \Re B_2'$

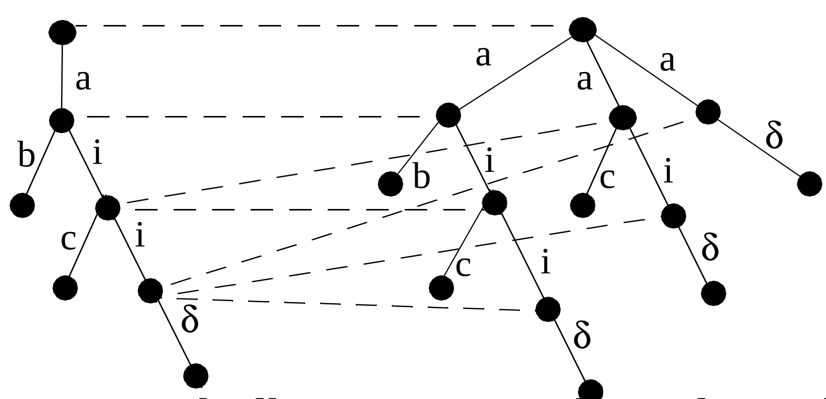
whenever $B_2 = s \Rightarrow B_2'$, then for some B_1':

$B_1 = s \Rightarrow B_1'$ and $B_1' \Re B_2'$

The idea of weak bisimulation is that two bisimilar LTS must be able to *simulate* each other, in terms of observable sequences, and then reach bisimilar LTS.

Definition

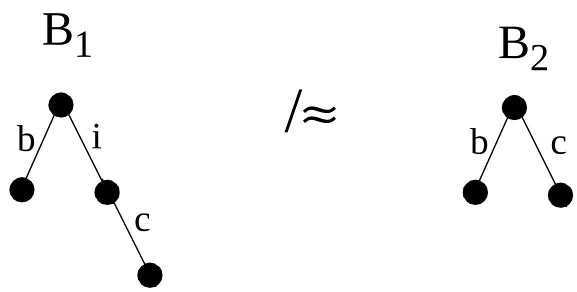
Two LTS B_1 and B_2 are *weak bisimulation equivalent* or *observationally equivalent* ($B_1 \approx B_2$) if there exists a weak bisimulation $\Re$ which contains the pair $\langle B_1, B_2 \rangle$.



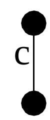
Note that a ai aii on the lhs go respectively to identical subrees as the three a alternatives on the rhs

BASIC LAW of O.E.: i;B ≈ B

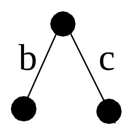
Basic case where B_1 /≈ B_2



Note:

B_1 = ε => 

B_1’

B_2 = ε => 

B_2’

Obviously B_1’ does not bisimulate B_2’ because B_2 = b => stop while the same thing is false for B_1’.

In some sense, observation equivalence can be considered *too strong* as it distinguishes between behavior trees that can be said to behave the same way externally (see trees III and VIII). For this reason, its name may be inappropriate.

However, o.e. has some desirable formal properties (in particular, it is defined inductively), so it is appropriate for formal reasoning.

Unfortunately, it is expensive to compute (it requires unbounded recursive descent in the LTS, while keeping track of $\mathfrak{R}$) so it is not often used in tools to simplify LTS.

Also, we shall now see another meaning of equivalence for which o.e. is in fact *too weak*...

CONTEXTS

A context C[•] is a behavior expression containing one or more occurrences of the special process variable [•] (the *hole*).

If B is a b. expr. then C[B] denotes the result of replacing [•] by B in context C[•].

Note then that 'the hole' represents a behavior expression, or , if we are talking about behavior trees, a whole behavior tree (from the root).

Observational Congruence

Two behaviours are *observationally congruent* when they can be interchanged one for the other in any behaviour expression, yielding *observationally equivalent* behaviour expressions.

More Formally:

Two b.exp. B_1 and B_2 are *observationally congruent* $B_1 \approx^c B_2$ iff for all contexts $C[\bullet]$, $C[B_1] \approx C[B_2]$.

Some Laws of Observation Equivalence and Congruence

Let C be a LOTOS context of the form

a;[•]	for these contexts
[•] |[A]| B or B |[A]| [•]	$\approx, \approx^c$
[•] >> B or B >> [•]	
[•] [> B	are the same
hide A in [•]	

Then $B_1 \approx B_2$ implies $C[B_1] \approx C[B_2]$

However in context a; **stop** [] [•]

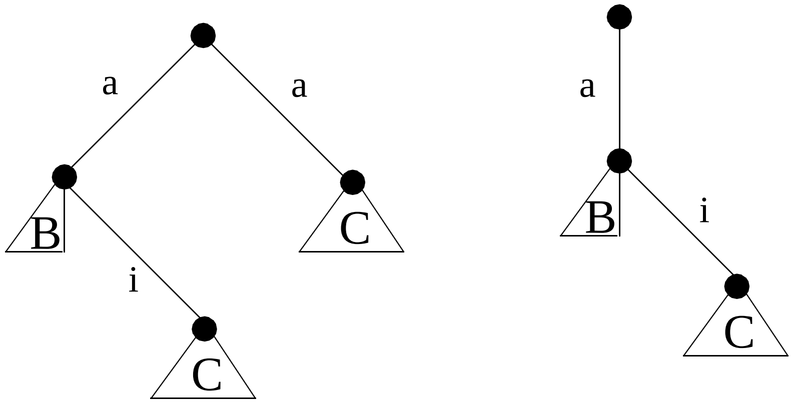
b;**stop** ≈ **i**;b;**stop**

But a;**stop**[]b;**stop** /≈ a; **stop** [] **i**;b;**stop**
(the devil can act only if there is a choice!)

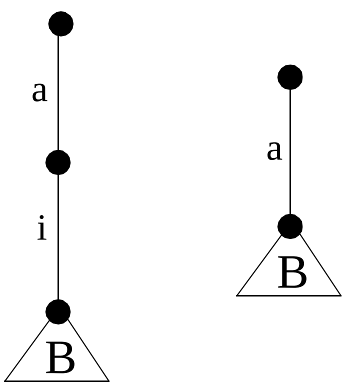
and similarly for context B [> [•]

(proof techniques to be discussed later)

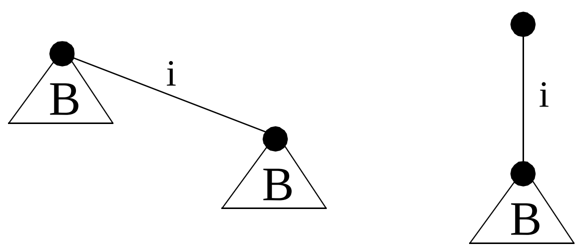
$a; (B \; [] \; \mathbf{i};C) \; [] \; a;C \approx^{c} a; (B \; [] \; \mathbf{i};C)$



$a; \mathbf{i}; B \approx^{c} a; B$



$B \; [] \; \mathbf{i}; B \approx^{c} \mathbf{i}; B$



Some Laws of Observational Congruence

Formal concepts of implementation: the intuitive basis (C.A.R.Hoare)

Suppose you have specified that you want a machine which

- always gives you coffee when you ask for it,
- but it may or may not give you tea (= can deadlock on tea, can refuse tea).

You must accept a machine which is only able to give you coffee!

This is because if you keep asking for tea, and not getting it, the maker can argue that this is consistent with your specification.

In LOTOS terms, the presence of internal actions in choices can be taken to represent implementation options.

Note that this reasoning breaks down in the presence of time constraints, which has interesting consequences for timed process algebras.

(non-symmetric)

RELATION *red* MODELS *IMPLEMENTATION* in the sense of *taking away unnecessary options*

I *red* S = (I *implements* S)

I can only execute actions that S can.
I can only refuse actions that S can.

E.g.

B *red* **i**;B [] **i**;C
C *red* **i**;B [] **i**;C
B *red* **i**;B [] C
note: C *red* **i**;B [] C is false
a;(B1[]B2) *red* a;B1 [] a;B2
note: converse is false
a;B1 *red* a;B1 [] a;B2

I *te* J = (I is *testing equivalent* to J) $=_{\text{def}}$
I *red* J and J *red* I

We will see that

$$= \subset \sim \subset \approx^{c} \subset \approx \subset te \subset tr$$

Also we will see that on the basis of these equivalence relations it is possible to

- define concepts of conformance and
- give algorithms for generating test suites to test whether an implementation conforms to a specification

Concepts discussed in Class 4:

• Importance of the concept of behavioral equivalence

• Why do we have several versions of this concept

• The most important equivalence concepts were introduced briefly

•• Trace equivalence
•• Identity or equality
•• Strong Bisimulation
•• Weak bisimulation or Observation equival.
•• Observation congruence
•• Reduction and testing equivalence

Will be discussed in detail...