The Linear Time – Branching Time Spectrum after 20 years

or

Full abstraction for safety and liveness properties

Rob van Glabbeek

NICTA, Sydney, Australia

University of New South Wales, Sydney, Australia

Celebration of 20 years of Concur
Bologna, 2nd September 2009

- If I had to pick just one semantic preorder with a good scope of useful applications, I'd say it should be the coarsest semantics that respects all safety and (conditional) liveness properties, and is compositional for hiding operators and parallel composition.

- If I had to pick just one semantic preorder with a good scope of useful applications, I'd say it should be the coarsest semantics that respects all safety and (conditional) liveness properties, and is compositional for hiding operators and parallel composition.
- This semantics has not be characterised before, so I contribute it here. It is very close in spirit to the may-and-must testing preorder of De Nicola & Hennessy, or the failures semantics of CSP, but I deviate from these works on 3 counts.

- If I had to pick just one semantic preorder with a good scope of useful applications, I'd say it should be the coarsest semantics that respects all safety and (conditional) liveness properties, and is compositional for hiding operators and parallel composition.
- This semantics has not be characterised before, so I contribute it here. It is very close in spirit to the may-and-must testing preorder of De Nicola & Hennessy, or the failures semantics of CSP, but I deviate from these works on 3 counts.

  There are three design decisions I would change.
  As these are orthogonal, I will present them one by one.

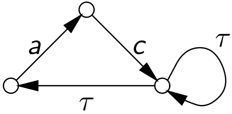
- If I had to pick just one semantic preorder with a good scope of useful applications, I'd say it should be the coarsest semantics that respects all safety and (conditional) liveness properties, and is compositional for hiding operators and parallel composition.
- This semantics has not be characterised before, so I contribute it here. It is very close in spirit to the may-and-must testing preorder of De Nicola & Hennessy, or the failures semantics of CSP, but I deviate from these works on 3 counts.

  There are three design decisions I would change.
  As these are orthogonal, I will present them one by one.
- Time permitting, at the end of my talk I will make some comments on the rest of the semantic lattice, and my current view on the linear time branching time spectrum.

Labelled Transition Systems

I focus on processes modelled as states in an LTS $(P, \rightarrow)$, where P is a set of states (or *processes*) and $\rightarrow \subseteq P \times Act_\tau \times P$ the *transition relation* for some set of *visible actions* Act augmented with the *invisible action* $\tau \notin Act$.

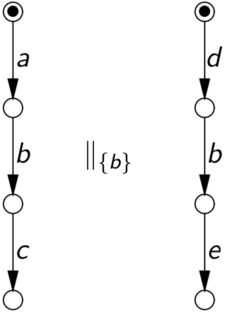
Thus, I abstract from probabilistic choice, real-time, etc.



Let $a, b, c, \ldots$ range over Act and $\alpha, \beta, \ldots$ over Act_τ.
An α-labelled transition from process (state) p to q is denoted $p \xrightarrow{\alpha} q$.

However, when explaining the difference between parallel composition and interleaving operators, I implicitly use Petri Nets or an enriched form of LTS as my system model.

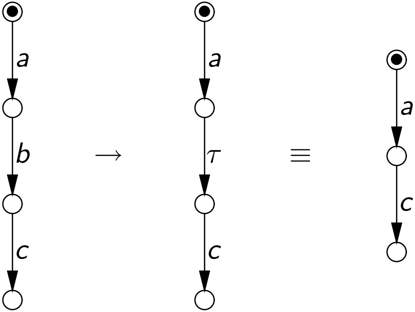
Partially synchronous parallel composition



as in CSP.

Hiding operators

Abstraction from the action b:



rename b into the hidden action τ.

Semantic equivalences

A useful semantic equivalence $\sim$ between processes (e.g. states in an LTS) has to satisfy two crucial requirements:

(1) Let Φ be the set of properties of processes that are important in applications.
If $p \sim q$ and p satisfies some property from Φ, then so does q.
In order words,
equivalent processes should have the same important properties.
Or,
if p has an important property that q does not have, they better be distinguished by $\sim$.

(2) If applications can be build by putting a process p in a context $C[p]$ (such as a parallel composition $p \| r$), then

$$p \sim q \;\Rightarrow\; C[p] \sim C[q].$$

Preorders

- Two crucial requirements of useful $\sim$:
  (1) respect important properties $\varphi \in \Phi$:
  $$p \sim q \;\Rightarrow\; p \models \varphi \Leftrightarrow q \models \varphi$$
  (2) compositionality (or congruence):
  $$p \sim q \;\Rightarrow\; C[p] \sim C[q].$$

Preorders

- Two crucial requirements of useful $\sim$:
  (1) respect important properties $\varphi \in \Phi$:
  $$p \sim q \;\Rightarrow\; p \models \varphi \Leftrightarrow q \models \varphi$$
  (2) compositionality (or congruence):
  $$p \sim q \;\Rightarrow\; C[p] \sim C[q].$$
- Following Hoare and De Nicola & Hennessy I write $S \sqsubseteq I$ if I is a correct implementation of the specification S.

Preorders

- Two crucial requirements of useful $\sim$:
  (1) respect important properties $\varphi \in \Phi$:
  $$p \sim q \;\Rightarrow\; p \models \varphi \Leftrightarrow q \models \varphi$$
  (2) compositionality (or congruence):
  $$p \sim q \;\Rightarrow\; C[p] \sim C[q].$$
- Following Hoare and De Nicola & Hennessy I write $S \sqsubseteq I$ if I is a correct implementation of the specification S.
- The crucial property (1) now becomes
  (1) $\sqsubseteq$ respects good properties $\varphi \in \Phi$:
  $$p \sqsubseteq q \;\Rightarrow\; p \models \varphi \Rightarrow q \models \varphi$$
  (Note: if φ is a good property, then $\neg\varphi$ is a bad property.)

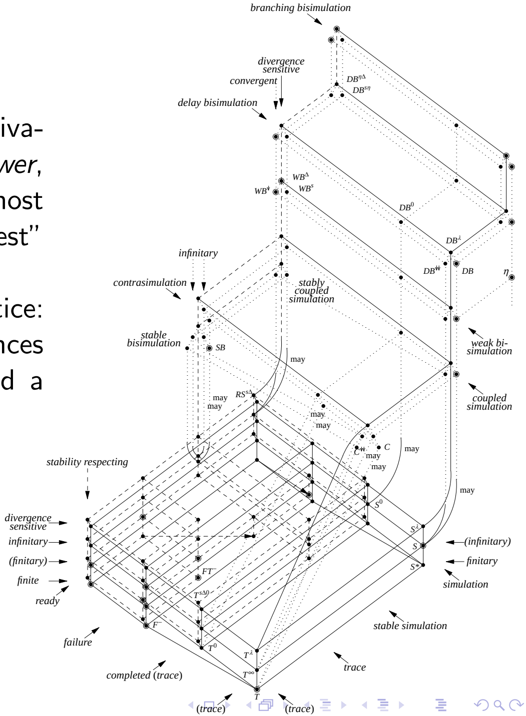
Preorders

- Two crucial requirements of useful $\sim$:
  (1) respect important properties $\varphi \in \Phi$:
  $$p \sim q \;\Rightarrow\; p \models \varphi \Leftrightarrow q \models \varphi$$
  (2) compositionality (or congruence):
  $$p \sim q \;\Rightarrow\; C[p] \sim C[q].$$
- Following Hoare and De Nicola & Hennessy I write $S \sqsubseteq I$ if I is a correct implementation of the specification S.
- The crucial property (1) now becomes
  (1) $\sqsubseteq$ respects good properties $\varphi \in \Phi$:
  $$p \sqsubseteq q \;\Rightarrow\; p \models \varphi \Rightarrow q \models \varphi$$
  (Note: if φ is a good property, then $\neg\varphi$ is a bad property.)
- The second property becomes monotonicity:
  $$p \sqsubseteq q \;\Rightarrow\; C[p] \sqsubseteq C[q].$$

The semantic lattice

We can order semantic equivalences by *distinguishing power*, drawing the “strongest”, “most discriminating”, or “finest” above.

They form a complete lattice: any collection of equivalences has a least upperbound and a greatest lowerbound.

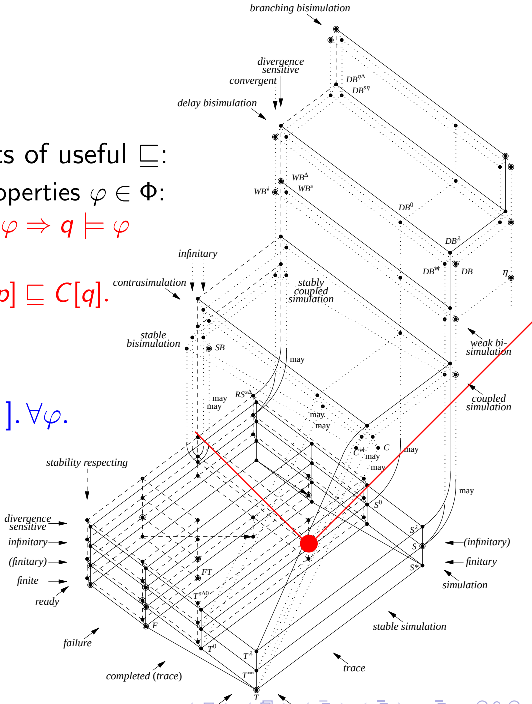


Full abstraction

- Two crucial requirements of useful $\sqsubseteq$:
  (1) respect important properties $\varphi \in \Phi$:
  $$p \sqsubseteq q \;\Rightarrow\; p \models \varphi \Rightarrow q \models \varphi$$
  (2) compositionality:
  $$p \sqsubseteq q \;\Rightarrow\; C[p] \sqsubseteq C[q].$$

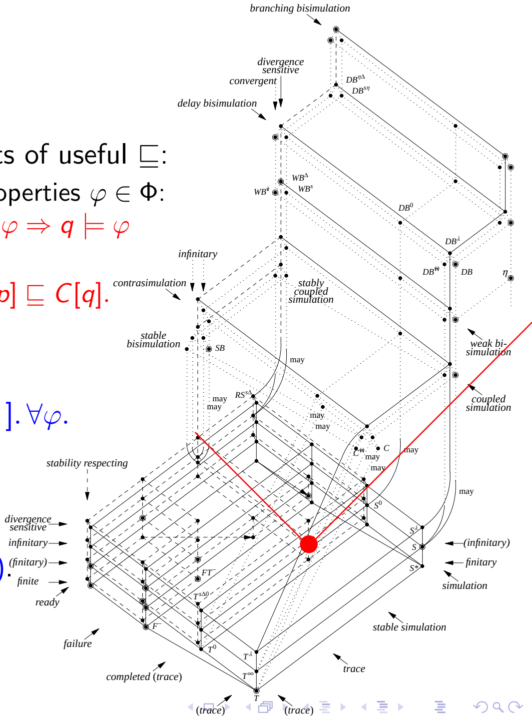
Full abstraction

- Two crucial requirements of useful $\sqsubseteq$:
  (1) respect important properties $\varphi \in \Phi$:
  $$p \sqsubseteq q \;\Rightarrow\; p \models \varphi \Rightarrow q \models \varphi$$
  (2) compositionality:
  $$p \sqsubseteq q \;\Rightarrow\; C[p] \sqsubseteq C[q].$$
- ●: coarsest preorder satisfying (1) and (2).
  Given by $p \sqsubseteq q \;\Leftrightarrow \forall C[\;].\forall\varphi.$
  $C[p] \models \varphi \Rightarrow C[q] \models \varphi.$
  Is called fully abstract.



Full abstraction

- Two crucial requirements of useful $\sqsubseteq$:
  (1) respect important properties $\varphi \in \Phi$:
  $$p \sqsubseteq q \;\Rightarrow\; p \models \varphi \Rightarrow q \models \varphi$$
  (2) compositionality:
  $$p \sqsubseteq q \;\Rightarrow\; C[p] \sqsubseteq C[q].$$
- ●: coarsest preorder satisfying (1) and (2). Given by $p \sqsubseteq q \;\Leftrightarrow \forall C[\;].\forall\varphi.$ $C[p] \models \varphi \Rightarrow C[q] \models \varphi.$ Is called fully abstract.
- above the red line: all satisfying (1) and (2).



Safety and liveness properties

The good properties to consider are

- Safety properties:

  Something bad will never happen

- Liveness properties:

  Something good will happen eventually

- Conditional liveness properties (to be explained).

Safety properties

- Let b be a special action, saying that something bad happens.
  A trace of a process p is the sequence of visible actions resulting from an execution starting in state p.
  Now my specific safety property says that a process has no trace in which the action b occurs. b will never happen

Safety properties

- Let b be a special action, saying that something bad happens.
  A trace of a process p is the sequence of visible actions resulting from an execution starting in state p.
  Now my specific safety property says that a process has no trace in which the action b occurs. b will never happen
- A general safety property is a set B of sequences of actions, thought of as all those traces that are bad for us, or make us unhappy for whatever reason.
  A process satisfies this general safety property iff it allows no traces in B.

Safety properties

- Let b be a special action, saying that something bad happens.
  A trace of a process p is the sequence of visible actions resulting from an execution starting in state p.
  Now my specific safety property says that a process has no trace in which the action b occurs. b will never happen
- A general safety property is a set B of sequences of actions, thought of as all those traces that are bad for us, or make us unhappy for whatever reason.
  A process satisfies this general safety property iff it allows no traces in B.
- Theorem: A congruence for hiding and parallel composition respects every general safety property iff it respects the specific safety property above.

Safety properties

- Let b be a special action, saying that something bad happens.
  A trace of a process p is the sequence of visible actions resulting from an execution starting in state p.
  Now my specific safety property says that a process has no trace in which the action b occurs. b will never happen
- A general safety property is a set B of sequences of actions, thought of as all those traces that are bad for us, or make us unhappy for whatever reason.
  A process satisfies this general safety property iff it allows no traces in B.
- Theorem: A congruence for hiding and parallel composition respects every general safety property iff it respects the specific safety property above.
- The preorder which is fully abstract w.r.t. safety properties and parallel composition and hiding is reverse trace inclusion:

$$p \sqsubseteq_{\text{safety}} q \Leftrightarrow \mathit{traces}(p) \supseteq \mathit{traces}(q)$$

Liveness properties

- Let g be a special action, saying that something good happens. A trace is *completed* it is the visible context of a *maximal* execution, that is either infinite, or ends in a *deadlock state*, from which no further transitions are possible.
  Now my specific liveness property says that in every completed trace of p the action g occurs. g will eventually happen

Liveness properties

- Let g be a special action, saying that something good happens.
  A trace is *completed* it is the visible context of a *maximal* execution, that is either infinite, or ends in a *deadlock state*, from which no further transitions are possible.
  Now my specific liveness property says that in every completed trace of p the action g occurs. g will eventually happen
- A general liveness property is a set G of sequences of actions, thought of as all those traces that are good for us, or make us happy for whatever reason.
  A process satisfies this general liveness property iff it allows only completed traces in G.

Liveness properties

- Let g be a special action, saying that something good happens.
  A trace is *completed* it is the visible context of a *maximal* execution, that is either infinite, or ends in a *deadlock state*, from which no further transitions are possible.
  Now my specific liveness property says that in every completed trace of p the action g occurs. g will eventually happen
- A general liveness property is a set G of sequences of actions, thought of as all those traces that are good for us, or make us happy for whatever reason.
  A process satisfies this general liveness property iff it allows only completed traces in G.
- Theorem: A congruence for hiding and parallel composition respects every general liveness property iff it respects the specific liveness property above.

Liveness properties

- Let g be a special action, saying that something good happens. A trace is *completed* it is the visible context of a *maximal* execution, that is either infinite, or ends in a *deadlock state*, from which no further transitions are possible.
  Now my specific liveness property says that in every completed trace of p the action g occurs. g will eventually happen
- A general liveness property is a set G of sequences of actions, thought of as all those traces that are good for us, or make us happy for whatever reason.
  A process satisfies this general liveness property iff it allows only completed traces in G.
- Theorem: A congruence for hiding and parallel composition respects every general liveness property iff it respects the specific liveness property above.
- The preorder which is fully abstract w.r.t. liveness properties and a form of parallel composition and hiding has been characterised by De Nicola & Hennessy as the must-testing preorder; it the CSP failures and divergences preorder.

May-testing versus safety preorder

- [DH84] defines the *may-testing* preorder $\sqsubseteq_{\mathrm{may}}$, which amounts to trace inclusion, and the *must-testing* preorder $\sqsubseteq_{\mathrm{must}}$. Then the combined testing preorder is given by

$$p \sqsubseteq_{\mathrm{testing}} q \text{ iff } p \sqsubseteq_{\mathrm{may}} q \text{ and } p \sqsubseteq_{\mathrm{must}} q.$$

May-testing versus safety preorder

- [DH84] defines the *may-testing* preorder $\sqsubseteq_{\mathrm{may}}$, which amounts to trace inclusion, and the *must-testing* preorder $\sqsubseteq_{\mathrm{must}}$. Then the combined testing preorder is given by

$$p \sqsubseteq_{\mathrm{testing}} q \text{ iff } p \sqsubseteq_{\mathrm{may}} q \text{ and } p \sqsubseteq_{\mathrm{must}} q.$$

- My program is essentially the same, but I deviate from the [DH84]-approach in 3 ways.

May-testing versus safety preorder

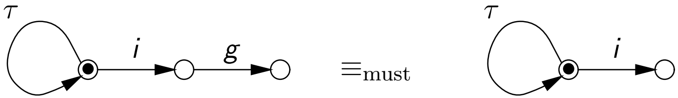
- [DH84] defines the *may-testing* preorder $\sqsubseteq_{\mathrm{may}}$, which amounts to trace inclusion, and the *must-testing* preorder $\sqsubseteq_{\mathrm{must}}$. Then the combined testing preorder is given by
$$p \sqsubseteq_{\mathrm{testing}} q \text{ iff } p \sqsubseteq_{\mathrm{may}} q \text{ and } p \sqsubseteq_{\mathrm{must}} q.$$
- My program is essentially the same, but I deviate from the [DH84]-approach in 3 ways.
- First of all, the safety preorder is the **reverse** of the may-testing preorder.
The process $ab + ac$ **may** do the action b. In may-testing semantics $ab + ac$ is a good implementation of ab, because everything that $ab + ac$ may do, also ab may do.
In the safety preorder, the ability to do b is a bad thing, which reverses the preorder.

May-testing versus safety preorder

- [DH84] defines the *may-testing* preorder $\sqsubseteq_{\rm may}$, which amounts to trace inclusion, and the *must-testing* preorder $\sqsubseteq_{\rm must}$. Then the combined testing preorder is given by
$$p \sqsubseteq_{\rm testing} q \text{ iff } p \sqsubseteq_{\rm may} q \text{ and } p \sqsubseteq_{\rm must} q.$$
- My program is essentially the same, but I deviate from the [DH84]-approach in 3 ways.
- First of all, the safety preorder is the **reverse** of the may-testing preorder.
  The process $ab + ac$ **may** do the action b. In may-testing semantics $ab + ac$ is a good implementation of ab, because everything that $ab + ac$ may do, also ab may do.
  In the safety preorder, the ability to do b is a bad thing, which reverses the preorder.
- Therefore, my first modification of the testing preorder is that I take $\sqsupseteq_{\rm may} \cap \sqsubseteq_{\rm must}$ rather than $\sqsubseteq_{\rm may} \cap \sqsubseteq_{\rm must}$.

The limits of must-testing

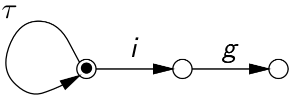
- In must-testing semantics these two processes are identified:



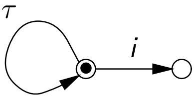
In neither case can we be sure that the good action g will eventually happen.
These two processes satisfy the same liveness properties.

The limits of must-testing

- In must-testing semantics these two processes are identified:



$\equiv_{\text{must}}$

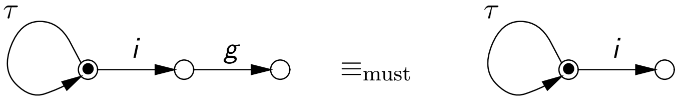


  In neither case can we be sure that the good action g will eventually happen.
  These two processes satisfy the same liveness properties.

- Now think of the action i as an investment, that costs us \$10.000, and of g as the investment paying off...

The limits of must-testing

- In must-testing semantics these two processes are identified:



  In neither case can we be sure that the good action g will eventually happen.
  These two processes satisfy the same liveness properties.
- Now think of the action i as an investment, that costs us \$10.000, and of g as the investment paying off...
- The must-testing preorder does not respect conditional liveness properties.

Conditional liveness properties

- Let i be an action that indicates a cost or investment, and g be the signal that this investment pays off.
  Now my specific liveness property says that in every completed trace of in which the action i occurs, g occurs as well. provided i occurs, g will eventually happen

Conditional liveness properties

- Let i be an action that indicates a cost or investment, and g be the signal that this investment pays off.
  Now my specific liveness property says that in every completed trace of in which the action i occurs, g occurs as well. provided i occurs, g will eventually happen
- A general conditional liveness property is a pair (σ, G) of a sequence of actions σ and a set of sequences of actions G.
  A process satisfies this general liveness property iff it allows only completed traces with the property that each completed trace with prefix σ occurs in G.

Conditional liveness properties

- Let i be an action that indicates a cost or investment, and g be the signal that this investment pays off.
  Now my specific liveness property says that in every completed trace of in which the action i occurs, g occurs as well. provided i occurs, g will eventually happen
- A general conditional liveness property is a pair (σ, G) of a sequence of actions σ and a set of sequences of actions G.
  A process satisfies this general liveness property iff it allows only completed traces with the property that each completed trace with prefix σ occurs in G.
- Theorem: A congruence for hiding and parallel composition respects every general conditional liveness property iff it respects the specific conditional liveness property above.

Conditional liveness properties

- Let i be an action that indicates a cost or investment, and g be the signal that this investment pays off.
  Now my specific liveness property says that in every completed trace of in which the action i occurs, g occurs as well. provided i occurs, g will eventually happen
- A general conditional liveness property is a function from the set of finite sequences of actions to the real numbers,
  This function indicates for each occurrence of an action in a complete trace how much profit or loss ones makes by executing this action, namely the value associated to the sequence of visible actions seen so far.
  A process satisfies this general liveness property iff each of its completed traces sums to a positive value. Reward testing

Conditional liveness properties

- Let i be an action that indicates a cost or investment, and g be the signal that this investment pays off.
  Now my specific liveness property says that in every completed trace of in which the action i occurs, g occurs as well. provided i occurs, g will eventually happen
- A general conditional liveness property is a function from the set of finite sequences of actions to the real numbers,
  This function indicates for each occurrence of an action in a complete trace how much profit or loss ones makes by executing this action, namely the value associated to the sequence of visible actions seen so far.
  A process satisfies this general liveness property iff each of its completed traces sums to a positive value. Reward testing
- Theorem: A congruence for hiding and parallel composition respects every general conditional liveness property iff it respects the specific conditional liveness property above.

Conditional liveness properties

- The preorder which is fully abstract w.r.t. these conditional liveness properties is also a congruence for normal liveness properties as well as safety properties.

Conditional liveness properties

- The preorder which is fully abstract w.r.t. these conditional liveness properties is also a congruence for normal liveness properties as well as safety properties.
- It coincides with the coarsest congruence respecting combined deadlock and divergence traces.

Conditional liveness properties

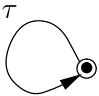
- The preorder which is fully abstract w.r.t. these conditional liveness properties is also a congruence for normal liveness properties as well as safety properties.
- It coincides with the coarsest congruence respecting combined deadlock and divergence traces.
- The latter has been characterised by Antti Puhakka: a process is determined by its:
  - divergence traces
  - eventually nondivergent infinite traces
  - and nondivergent failure pairs.

Conditional liveness properties

- The preorder which is fully abstract w.r.t. these conditional liveness properties is also a congruence for normal liveness properties as well as safety properties.
- It coincides with the coarsest congruence respecting combined deadlock and divergence traces.
- The latter has been characterised by Antti Puhakka: a process is determined by its:
  - divergence traces
  - eventually nondivergent infinite traces
  - and nondivergent failure pairs.
- $p \sqsubseteq_{\text{cond. liveness}} q \Leftrightarrow \begin{array}{l} \mathit{div.traces}(p) \supseteq \mathit{divtraces}(q) \\ \mathit{e.nd.inf.tr}(p) \supseteq \mathit{e.nd.inf.tr}(q) \\ \mathit{nd.fail}(p) \supseteq \mathit{nd.fail}(q) \;. \end{array}$

Parallel composition versus interleaving

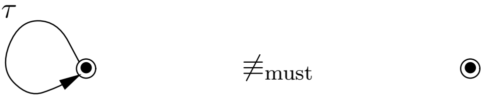
- In [DH84] must-testing semantics, or in [BHR84] failures semantics, livelock and deadlock are distinguished:



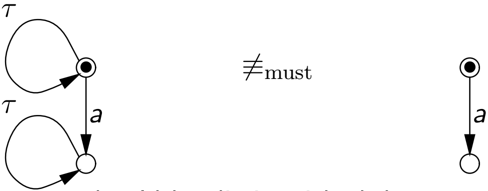
$\neq_{\text{must}}$

Parallel composition versus interleaving

- In [DH84] must-testing semantics, or in [BHR84] failures semantics, livelock and deadlock are distinguished:



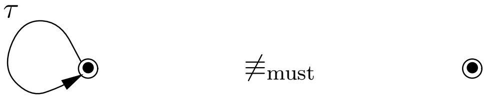
- The reason: when interleaving both process with the process a we get:



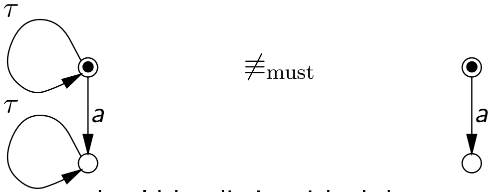
and those processes should be distinguished, because only the second one will certainly do an a.

Parallel composition versus interleaving

- In [DH84] must-testing semantics, or in [BHR84] failures semantics, livelock and deadlock are distinguished:



- The reason: when interleaving both process with the process a we get:



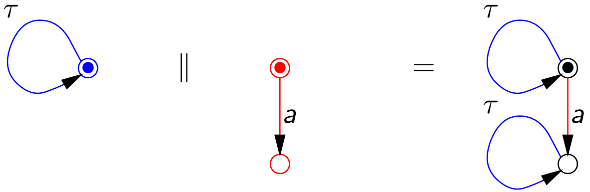
  and those processes should be distinguished, because only the second one will certainly do an a.

- It is not possibly to distinguish the original two processes when using parallel composition instead of interleaving!

Must-testing without interleaving

Must-testing without interleaving

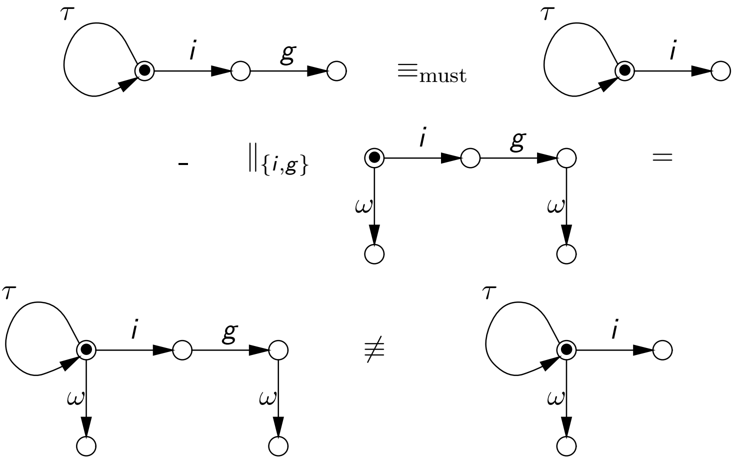
- To define testing semantics with parallel composition instead of interleaving, we take any model of concurrency that administers which transitions originates from the same component in a parallel composition. A trace now counts as completed only if it is completed in each parallel component.



This parallel composition does not have a completed trace without the a action.

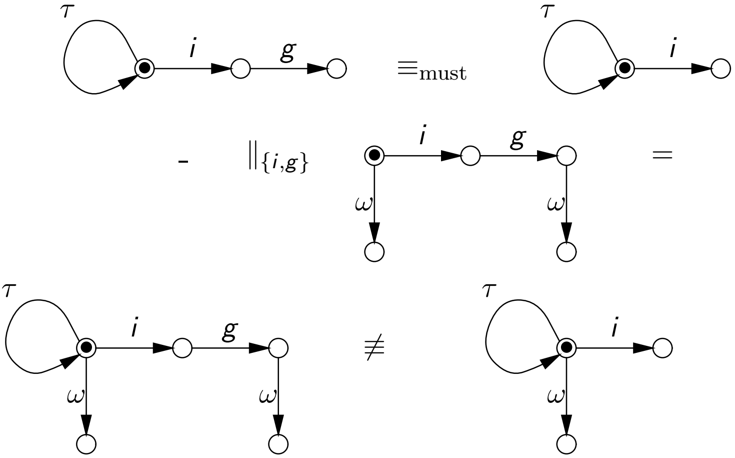
Must-testing with parallel comp. implies cond. liveness

- These two processes were identified in must-testing semantics, although they are distinguished by a cond. liveness property:



Must-testing with parallel comp. implies cond. liveness

- These two processes were identified in must-testing semantics, although they are distinguished by a cond. liveness property:



- On sequential processes resulting preorder exactly as before, but with extra identification of deadlock and livelock.

Conclusions / Position statement

- A useful semantic equivalence
  - respects important properties of processes
  - is compositional w.r.t important composition operators

Conclusions / Position statement

- A useful semantic equivalence
  - respects important properties of processes
  - is compositional w.r.t important composition operators
- Given agreement on these properties and operators, the above two requirements determine a unique best semantics
  This semantics is fully abstract w.r.t. the given collection of properties and operators.

Conclusions / Position statement

- A useful semantic equivalence
  - respects important properties of processes
  - is compositional w.r.t important composition operators
- Given agreement on these properties and operators, the above two requirements determine a unique best semantics
  This semantics is fully abstract w.r.t. the given collection of properties and operators.
- The role of other requirements, such as
  - complexity of decision procedures
  - good algebraic properties
    guarded fixed point equations have unique solutions

  is debatable.

Conclusions / Position statement

- A useful semantic equivalence
  - respects important properties of processes
  - is compositional w.r.t important composition operators
- Given agreement on these properties and operators, the above two requirements determine a unique best semantics
  This semantics is fully abstract w.r.t. the given collection of properties and operators.
- The role of other requirements, such as
  - complexity of decision procedures
  - good algebraic properties
    guarded fixed point equations have unique solutions

  is debatable.
- Different applications require different properties and operators; there is no canonical choice.

Conclusions / Position statement

- A useful semantic equivalence
  - respects important properties of processes
  - is compositional w.r.t important composition operators
- Given agreement on these properties and operators, the above two requirements determine a unique best semantics
  This semantics is fully abstract w.r.t. the given collection of properties and operators.
- The role of other requirements, such as
  - complexity of decision procedures
  - good algebraic properties
    guarded fixed point equations have unique solutions

  is debatable.
- Different applications require different properties and operators; there is no canonical choice.
- In the absence of agreement on which properties and operators to use, the finest (branching time) semantics are best.

Conclusions / Position statement

- A good semantics should respect the following properties
  - Safety properties
  - Liveness properties (possibly assuming fairness)
  - Conditional liveness properties (possibly assuming fairness)
  - (perhaps) AGEF properties

  Many other properties, such as preservation of deadlock behaviour, are not really important.

Conclusions / Position statement

- A good semantics should respect the following properties
  - Safety properties
  - Liveness properties (possibly assuming fairness)
  - Conditional liveness properties (possibly assuming fairness)
  - (perhaps) AGEF properties

  Many other properties, such as preservation of deadlock behaviour, are not really important.
- Compositionality is often required for
  - abstraction from internal activity
  - (partially synchronous) interleaving operator

  I believe it makes sense to use a (partially synchronous) parallel composition instead (while employing interleaving semantics by abstracting from causality etc).

Conclusions / Position statement

- A good semantics should respect the following properties
  - Safety properties
  - Liveness properties (possibly assuming fairness)
  - Conditional liveness properties (possibly assuming fairness)
  - (perhaps) AGEF properties

  Many other properties, such as preservation of deadlock behaviour, are not really important.
- Compositionality is often required for
  - abstraction from internal activity
  - (partially synchronous) interleaving operator

  I believe it makes sense to use a (partially synchronous) parallel composition instead (while employing interleaving semantics by abstracting from causality etc).
- Other operators may be needed depending on the application. A good example are priority operators.

Conclusions / Position statement

- May-testing equivalence is fully abstract for safety properties
- Must testing is fully abstract for liveness properties (w.r.t. abstraction and interleaving) De Nicola & Hennessy 1984

Conclusions / Position statement

- May-testing equivalence is fully abstract for safety properties
- Must testing is fully abstract for liveness properties (w.r.t. abstraction and interleaving) De Nicola & Hennessy 1984
- The may-testing preorder as stated is not particularly useful; What we need is its inverse:

  it is fully abstract for safety properties .

Conclusions / Position statement

- May-testing equivalence is fully abstract for safety properties
- Must testing is fully abstract for liveness properties (w.r.t. abstraction and interleaving) De Nicola & Hennessy 1984
- The may-testing preorder as stated is not particularly useful; What we need is its inverse:

  it is fully abstract for safety properties .
- The must-testing preorder is not strong enough.
  It is fully abstract for liveness properties but misses out on conditional liveness properties, which are just as important.

Conclusions / Position statement

- May-testing equivalence is fully abstract for safety properties
- Must testing is fully abstract for liveness properties (w.r.t. abstraction and interleaving) De Nicola & Hennessy 1984
- The may-testing preorder as stated is not particularly useful; What we need is its inverse:

  it is fully abstract for safety properties .
- The must-testing preorder is not strong enough. It is fully abstract for liveness properties but misses out on conditional liveness properties, which are just as important.
- I presented a finer semantics that is fully abstract for safety and conditional liveness properties w.r.t. abstraction and interleaving.

Conclusions / Position statement

- May-testing equivalence is fully abstract for safety properties
- Must testing is fully abstract for liveness properties (w.r.t. abstraction and interleaving) De Nicola & Hennessy 1984
- The may-testing preorder as stated is not particularly useful; What we need is its inverse:

  it is fully abstract for safety properties .
- The must-testing preorder is not strong enough. It is fully abstract for liveness properties but misses out on conditional liveness properties, which are just as important.
- I presented a finer semantics that is fully abstract for safety and conditional liveness properties w.r.t. abstraction and interleaving.
- At least two kinds of applications call for finer preorders:

Conclusions / Position statement

- May-testing equivalence is fully abstract for safety properties
- Must testing is fully abstract for liveness properties (w.r.t. abstraction and interleaving) De Nicola & Hennessy 1984
- The may-testing preorder as stated is not particularly useful; What we need is its inverse:
  it is fully abstract for safety properties .
- The must-testing preorder is not strong enough.
  It is fully abstract for liveness properties but misses out on conditional liveness properties, which are just as important.
- I presented a finer semantics that is fully abstract for safety and conditional liveness properties w.r.t. abstraction and interleaving.
- At least two kinds of applications call for finer preorders:
  - Priority calls for ready-trace (or failure-trace) semantics.

Conclusions / Position statement

- May-testing equivalence is fully abstract for safety properties
- Must testing is fully abstract for liveness properties (w.r.t. abstraction and interleaving) De Nicola & Hennessy 1984
- The may-testing preorder as stated is not particularly useful; What we need is its inverse:

  it is fully abstract for safety properties .
- The must-testing preorder is not strong enough.
  It is fully abstract for liveness properties but misses out on conditional liveness properties, which are just as important.
- I presented a finer semantics that is fully abstract for safety and conditional liveness properties w.r.t. abstraction and interleaving.
- At least two kinds of applications call for finer preorders:
  - Priority calls for ready-trace (or failure-trace) semantics.
  - Probabilistic contexts pushes us up into the branching time side of the spectrum: the failure simulation preorder.

Conclusions / Position statement

- Due to the use of interleaving operators, must-testing and related semantics make distinctions that are wholly unobservable when using merely parallel composition.

Conclusions / Position statement

- Due to the use of interleaving operators, must-testing and related semantics make distinctions that are wholly unobservable when using merely parallel composition.
- I propose a new semantics that is fully abstract for safety and (conditional) liveness properties w.r.t. hiding and parallel composition.