Formal Techniques for Software Engineering: Modal Logics

Rocco De Nicola

IMT Institute for Advanced Studies, Lucca
rocco.denicola@imtlucca.it

June 2013

Lesson 12

Verifying Correctness of Reactive Systems

Let *Impl* be an implementation of a system (e.g. in CCS syntax).

Equivalence Checking Approach

$$Impl \equiv Spec$$

- $\equiv$ is an abstract equivalence, e.g. $\sim$ or $\approx$
- *Spec* is often expressed in the same language as *Impl*
- *Spec* provides the full specification of the intended behaviour

Model Checking Approach

$$Impl \models Property$$

- $\models$ is the satisfaction relation
- *Property* is a particular feature, often expressed via a logic
- *Property* is a partial specification of the intended behaviour

Verifying Correctness of Reactive Systems

Let *Impl* be an implementation of a system (e.g. in CCS syntax).

Equivalence Checking Approach

$$Impl \equiv Spec$$

- $\equiv$ is an abstract equivalence, e.g. $\sim$ or $\approx$
- *Spec* is often expressed in the same language as *Impl*
- *Spec* provides the full specification of the intended behaviour

Model Checking Approach

$$Impl \models Property$$

- $\models$ is the satisfaction relation
- *Property* is a particular feature, often expressed via a logic
- *Property* is a partial specification of the intended behaviour

Model Checking of Reactive Systems

Our Aim

Develop a logic in which we can express interesting properties of reactive systems.

Logical Properties of Reactive Systems

Modal Properties – what can happen now (possibility, necessity)

- drink a coffee (can drink a coffee now)
- does not drink tea
- drinks both tea and coffee
- drinks tea after coffee

Temporal Properties – behaviour in time

- never drinks any alcohol
  (safety property: nothing bad can happen)
- eventually will have a glass of wine
  (liveness property: something good will happen)

Can these properties be expressed using equivalence checking?

Logical Properties of Reactive Systems

Modal Properties – what can happen now (possibility, necessity)

- drink a coffee (can drink a coffee now)
- does not drink tea
- drinks both tea and coffee
- drinks tea after coffee

Temporal Properties – behaviour in time

- never drinks any alcohol
  (safety property: nothing bad can happen)
- eventually will have a glass of wine
  (liveness property: something good will happen)

Can these properties be expressed using equivalence checking?

Logical Properties of Reactive Systems

Modal Properties – what can happen now (possibility, necessity)

- drink a coffee (can drink a coffee now)
- does not drink tea
- drinks both tea and coffee
- drinks tea after coffee

Temporal Properties – behaviour in time

- never drinks any alcohol
  (safety property: nothing bad can happen)
- eventually will have a glass of wine
  (liveness property: something good will happen)

Can these properties be expressed using equivalence checking?

Hennessy-Milner Logic – Syntax

Syntax of the Formulae ($a \in Act$)

$$F, G \ ::= \ tt \ | \ ff \ | \ F \wedge G \ | \ F \vee G \ | \ \langle a \rangle F \ | \ [a]F$$

- tt all processes satisfy this property
- ff no process satisfies this property
- $\wedge$, $\vee$ usual logical AND and OR
- $\langle a \rangle F$ (possibility) asserts (of a given P): It is possible for P to perform an action a and evolve into a Q that satisfies F - there is at least one a-successor that satisfies F
- $[a]F$ (necessity) asserts (of a given P): If P can perform an action a then it must evolve into a Q that satisfies F - all a-successors have to satisfy F

- $\langle a \rangle tt$ expresses the capability of performing action a.
- $[a]ff$ expresses the inability to perform an action a.

Hennessy-Milner Logic – Syntax

Syntax of the Formulae ($a \in Act$)

$$F, G \ ::= \ tt \ | \ ff \ | \ F \wedge G \ | \ F \vee G \ | \ \langle a \rangle F \ | \ [a]F$$

tt all processes satisfy this property

ff no process satisfies this property

$\wedge$, $\vee$ usual logical AND and OR

$\langle a \rangle F$ (possibility) asserts (of a given P): It is possible for P to perform an action a and evolve into a Q that satisfies F - there is at least one a-successor that satisfies F

$[a]F$ (necessity) asserts (of a given P): If P can perform an action a then it must evolve into a Q that satisfies F - all a-successors have to satisfy F

- $\langle a \rangle tt$ expresses the capability of performing action a.
- $[a]ff$ expresses the inability to perform an action a.

Hennessy-Milner Logic – Syntax

Syntax of the Formulae ($a \in Act$)

$$F, G \;::=\; tt \mid ff \mid F \wedge G \mid F \vee G \mid \langle a \rangle F \mid [a]F$$

- tt all processes satisfy this property
- ff no process satisfies this property
- $\wedge$, $\vee$ usual logical AND and OR
- $\langle a \rangle F$ (possibility) asserts (of a given P): It is possible for P to perform an action a and evolve into a Q that satisfies F - there is at least one a-successor that satisfies F
- $[a]F$ (necessity) asserts (of a given P): If P can perform an action a then it must evolve into a Q that satisfies F - all a-successors have to satisfy F

- $\langle a \rangle tt$ expresses the capability of performing action a.
- $[a]ff$ expresses the inability to perform an action a.

Hennessy-Milner Logic – Syntax

Syntax of the Formulae ($a \in Act$)

$$F, G \;::=\; tt \mid ff \mid F \wedge G \mid F \vee G \mid \langle a \rangle F \mid [a]F$$

- tt all processes satisfy this property
- ff no process satisfies this property
- $\wedge$, $\vee$ usual logical AND and OR
- $\langle a \rangle F$ (possibility) asserts (of a given P): It is possible for P to perform an action a and evolve into a Q that satisfies F - there is at least one a-successor that satisfies F
- $[a]F$ (necessity) asserts (of a given P): If P can perform an action a then it must evolve into a Q that satisfies F - all a-successors have to satisfy F

- $\langle a \rangle tt$ expresses the capability of performing action a.
- $[a]ff$ expresses the inability to perform an action a.

Hennessy-Milner Logic – Syntax

Syntax of the Formulae ($a \in Act$)

$$F, G \ ::= \ tt \ | \ ff \ | \ F \wedge G \ | \ F \vee G \ | \ \langle a \rangle F \ | \ [a]F$$

- tt all processes satisfy this property
- ff no process satisfies this property
- $\wedge$, $\vee$ usual logical AND and OR
- $\langle a \rangle F$ (possibility) asserts (of a given P): It is possible for P to perform an action a and evolve into a Q that satisfies F - there is at least one a-successor that satisfies F
- $[a]F$ (necessity) asserts (of a given P): If P can perform an action a then it must evolve into a Q that satisfies F - all a-successors have to satisfy F

- $\langle a \rangle tt$ expresses the capability of performing action a.
- $[a]ff$ expresses the inability to perform an action a.

Hennessy-Milner Logic – Semantics

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

Validity of the logical triple $p \models F$ ($p \in Proc$, F a HM formula)

$p \models tt$ for each $p \in Proc$

$p \models ff$ for no p (we also write $p \not\models ff$)

$p \models F \wedge G$ iff $p \models F$ and $p \models G$

$p \models F \vee G$ iff $p \models F$ or $p \models G$

$p \models \langle a \rangle F$ iff $p \xrightarrow{a} p'$ for some $p' \in Proc$ such that $p' \models F$

$p \models [a]F$ iff $p' \models F$, for all $p' \in Proc$ such that $p \xrightarrow{a} p'$

We write:

- $p \not\models F$ if p does not satisfy F
- $\langle \{a_1, a_2, \ldots a_n\} \rangle F$ for $\langle a_1 \rangle F \vee \langle a_2 \rangle F \cdots \vee \langle a_n \rangle F$
- $[\{a_1, a_2, \ldots a_n\}]F$ for $[a_1]F \wedge [a_2]F \cdots \wedge [a_n]F$

Hennessy-Milner Logic – Semantics

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

Validity of the logical triple $p \models F$ ($p \in Proc$, F a HM formula)

$p \models tt$ for each $p \in Proc$

$p \models ff$ for no p (we also write $p \not\models ff$)

$p \models F \wedge G$ iff $p \models F$ and $p \models G$

$p \models F \vee G$ iff $p \models F$ or $p \models G$

$p \models \langle a \rangle F$ iff $p \xrightarrow{a} p'$ for some $p' \in Proc$ such that $p' \models F$

$p \models [a]F$ iff $p' \models F$, for all $p' \in Proc$ such that $p \xrightarrow{a} p'$

We write:

- $p \not\models F$ if p does not satisfy F
- $\langle\{a_1, a_2, \dots a_n\}\rangle F$ for $\langle a_1 \rangle F \vee \langle a_2 \rangle F \cdots \vee \langle a_n \rangle F$
- $[\{a_1, a_2, \dots a_n\}]F$ for $[a_1]F \wedge [a_2]F \cdots \wedge [a_n]F$

Hennessy-Milner Logic – Semantics

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

Validity of the logical triple $p \models F$ ($p \in Proc$, F a HM formula)

$p \models tt$ for each $p \in Proc$

$p \models ff$ for no p (we also write $p \not\models ff$)

$p \models F \wedge G$ iff $p \models F$ and $p \models G$

$p \models F \vee G$ iff $p \models F$ or $p \models G$

$p \models \langle a \rangle F$ iff $p \xrightarrow{a} p'$ for some $p' \in Proc$ such that $p' \models F$

$p \models [a]F$ iff $p' \models F$, for all $p' \in Proc$ such that $p \xrightarrow{a} p'$

We write:

- $p \not\models F$ if p does not satisfy F
- $\langle \{a_1, a_2, \ldots a_n\} \rangle F$ for $\langle a_1 \rangle F \vee \langle a_2 \rangle F \cdots \vee \langle a_n \rangle F$
- $[\{a_1, a_2, \ldots a_n\}]F$ for $[a_1]F \wedge [a_2]F \cdots \wedge [a_n]F$

Hennessy-Milner Logic – Semantics

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

Validity of the logical triple $p \models F$ ($p \in Proc$, F a HM formula)

$p \models tt$ for each $p \in Proc$

$p \models ff$ for no p (we also write $p \not\models ff$)

$p \models F \wedge G$ iff $p \models F$ and $p \models G$

$p \models F \vee G$ iff $p \models F$ or $p \models G$

$p \models \langle a \rangle F$ iff $p \xrightarrow{a} p'$ for some $p' \in Proc$ such that $p' \models F$

$p \models [a]F$ iff $p' \models F$, for all $p' \in Proc$ such that $p \xrightarrow{a} p'$

We write:

- $p \not\models F$ if p does not satisfy F
- $\langle \{a_1, a_2, \ldots a_n\} \rangle F$ for $\langle a_1 \rangle F \vee \langle a_2 \rangle F \cdots \vee \langle a_n \rangle F$
- $[\{a_1, a_2, \ldots a_n\}]F$ for $[a_1]F \wedge [a_2]F \cdots \wedge [a_n]F$

Examples

- $E \models < tick > tt$
  E can do a tick
- $E \models < tick >< tock > tt$
  E can do a tick and then a tock
- $E \models < \{tick, tock\} > tt$
  E can do a tick or a tock
- $E \models \ [tick]ff$
  E cannot do a tick
- $E \models < tick > ff$
  This is equivalent to false!
- $E \models \ [tick]tt$
  This is equivalent to true!

Examples

- $E \models < tick > tt$
  E can do a tick
- $E \models < tick >< tock > tt$
  E can do a tick and then a tock
- $E \models < \{tick, tock\} > tt$
  E can do a tick or a tock
- $E \models [tick]ff$
  E cannot do a tick
- $E \models < tick > ff$
  This is equivalent to false!
- $E \models [tick]tt$
  This is equivalent to true!

Examples

- $E \models < tick > tt$
  E can do a tick
- $E \models < tick >< tock > tt$
  E can do a tick and then a tock
- $E \models < \{tick, tock\} > tt$
  E can do a tick or a tock
- $E \models [tick]ff$
  E cannot do a tick
- $E \models < tick > ff$
  This is equivalent to false!
- $E \models [tick]tt$
  This is equivalent to true!

Examples

- $E \models < tick > tt$
  E can do a tick
- $E \models < tick >< tock > tt$
  E can do a tick and then a tock
- $E \models < \{tick, tock\} > tt$
  E can do a tick or a tock
- $E \models [tick]ff$
  E cannot do a tick
- $E \models < tick > ff$
  This is equivalent to false!
- $E \models [tick]tt$
  This is equivalent to true!

Examples

- $E \models < tick > tt$
  E can do a tick
- $E \models < tick >< tock > tt$
  E can do a tick and then a tock
- $E \models < \{tick, tock\} > tt$
  E can do a tick or a tock
- $E \models \ [tick]ff$
  E cannot do a tick
- $E \models < tick > ff$
  This is equivalent to false!
- $E \models \ [tick]tt$
  This is equivalent to true!

Examples

- $E \models < tick > tt$
  E can do a tick
- $E \models < tick >< tock > tt$
  E can do a tick and then a tock
- $E \models < \{tick, tock\} > tt$
  E can do a tick or a tock
- $E \models \ [tick]ff$
  E cannot do a tick
- $E \models < tick > ff$
  This is equivalent to false!
- $E \models \ [tick]tt$
  This is equivalent to true!

Checking satisfaction

$C1 =_{def} tick.C1$

Does $C1$ have property: $[tick](< tick > tt \wedge [tock]ff)$?

$$C1 \models [tick](< tick > tt \wedge [tock]ff)$$

iff $\forall F \in \{E : C1 \xrightarrow{tick} E\}.\ F \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt$ *and* $C1 \models [tock]ff$

iff $\exists F \in \{E : C1 \xrightarrow{tick} E\}$ *and* $C1 \models [tock]ff$

iff $\exists F \in \{C1\}$ *and* $C1 \models [tock]ff$

iff $C1 \models [tock]ff$

iff $\{E : C1 \xrightarrow{tock} E\} = \emptyset$

Checking satisfaction

$C1 =_{def} tick.C1$

Does $C1$ have property: $[tick](< tick > tt \wedge [tock]ff)$?

$$C1 \models [tick](< tick > tt \wedge [tock]ff)$$

iff $\forall F \in \{E : C1 \xrightarrow{tick} E\}.\ F \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt$ and $C1 \models [tock]ff$

iff $\exists F \in \{E : C1 \xrightarrow{tick} E\}$ and $C1 \models [tock]ff$

iff $\exists F \in \{C1\}$ and $C1 \models [tock]ff$

iff $C1 \models [tock]ff$

iff $\{E : C1 \xrightarrow{tock} E\} = \emptyset$

Checking satisfaction

$C1 =_{def} tick.C1$

Does $C1$ have property: $[tick](< tick > tt \wedge [tock]ff)$?

$$C1 \models [tick](< tick > tt \wedge [tock]ff)$$

iff $\forall F \in \{E : C1 \stackrel{tick}{\longrightarrow} E\}.\ F \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt$ and $C1 \models [tock]ff$

iff $\exists F \in \{E : C1 \stackrel{tick}{\longrightarrow} E\}$ and $C1 \models [tock]ff$

iff $\exists F \in \{C1\}$ and $C1 \models [tock]ff$

iff $C1 \models [tock]ff$

iff $\{E : C1 \stackrel{tock}{\longrightarrow} E\} = \emptyset$

Checking satisfaction

$C1 =_{def} tick.C1$

Does $C1$ have property: $[tick](< tick > tt \wedge [tock]ff)$?

$$C1 \models [tick](< tick > tt \wedge [tock]ff)$$

iff $\forall F \in \{E : C1 \xrightarrow{tick} E\}.\ F \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt$ and $C1 \models [tock]ff$

iff $\exists F \in \{E : C1 \xrightarrow{tick} E\}$ and $C1 \models [tock]ff$

iff $\exists F \in \{C1\}$ and $C1 \models [tock]ff$

iff $C1 \models [tock]ff$

iff $\{E : C1 \xrightarrow{tock} E\} = \emptyset$

Checking satisfaction

$C1 =_{def} tick.C1$

Does $C1$ have property: $[tick](< tick > tt \wedge [tock]ff)$?

$$C1 \models [tick](< tick > tt \wedge [tock]ff)$$

iff $\forall F \in \{E : C1 \xrightarrow{tick} E\}.\ F \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt$ *and* $C1 \models [tock]ff$

iff $\exists F \in \{E : C1 \xrightarrow{tick} E\}$ *and* $C1 \models [tock]ff$

iff $\exists F \in \{C1\}$ *and* $C1 \models [tock]ff$

iff $C1 \models [tock]ff$

iff $\{E : C1 \xrightarrow{tock} E\} = \emptyset$

Checking satisfaction

$C1 =_{def} tick.C1$

Does $C1$ have property: $[tick](< tick > tt \wedge [tock]ff)$?

$$C1 \models [tick](< tick > tt \wedge [tock]ff)$$

iff $\forall F \in \{E : C1 \xrightarrow{tick} E\}.\ F \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt \wedge [tock]ff$

iff $C1 \models < tick > tt$ *and* $C1 \models [tock]ff$

iff $\exists F \in \{E : C1 \xrightarrow{tick} E\}$ *and* $C1 \models [tock]ff$

iff $\exists F \in \{C1\}$ *and* $C1 \models [tock]ff$

iff $C1 \models [tock]ff$

iff $\{E : C1 \xrightarrow{tock} E\} = \emptyset$

Checking satisfaction

$C1 =_{def} tick.C1$

Does $C1$ have property: $[tick](< tick > tt \wedge [tock]ff)$?

$$C1 \models [tick](< tick > tt \wedge [tock]ff)$$

$$\text{iff } \forall F \in \{E : C1 \xrightarrow{tick} E\}.\ F \models < tick > tt \wedge [tock]ff$$

$$\text{iff } C1 \models < tick > tt \wedge [tock]ff$$

$$\text{iff } C1 \models < tick > tt \text{ and } C1 \models [tock]ff$$

$$\text{iff } \exists F \in \{E : C1 \xrightarrow{tick} E\} \text{ and } C1 \models [tock]ff$$

$$\text{iff } \exists F \in \{C1\} \text{ and } C1 \models [tock]ff$$

$$\text{iff } C1 \models [tock]ff$$

$$\text{iff } \{E : C1 \xrightarrow{tock} E\} = \emptyset$$

Checking satisfaction

$C1 =_{def} tick.C1$

Does $C1$ have property: $[tick](< tick > tt \wedge [tock]ff)$?

$$C1 \models [tick](< tick > tt \wedge [tock]ff)$$

$$\text{iff } \forall F \in \{E : C1 \xrightarrow{tick} E\}.\ F \models < tick > tt \wedge [tock]ff$$

$$\text{iff } C1 \models < tick > tt \wedge [tock]ff$$

$$\text{iff } C1 \models < tick > tt \textit{ and } C1 \models [tock]ff$$

$$\text{iff } \exists F \in \{E : C1 \xrightarrow{tick} E\} \textit{ and } C1 \models [tock]ff$$

$$\text{iff } \exists F \in \{C1\} \textit{ and } C1 \models [tock]ff$$

$$\text{iff } C1 \models [tock]ff$$

$$\text{iff } \{E : C1 \xrightarrow{tock} E\} = \emptyset$$

What about Negation?

For every formula F we define the formula F^c as follows:

- $tt^c = ff$
- $ff^c = tt$
- $(F \wedge G)^c = F^c \vee G^c$
- $(F \vee G)^c = F^c \wedge G^c$
- $(\langle a \rangle F)^c = [a]F^c$
- $([a]F)^c = \langle a \rangle F^c$

Theorem (F^c is equivalent to the negation of F)

For any $p \in Proc$ and any HML formula F

1. $p \models F \Longrightarrow p \not\models F^c$
2. $p \not\models F \Longrightarrow p \models F^c$

What about Negation?

For every formula F we define the formula F^c as follows:

- $tt^c = ff$
- $ff^c = tt$
- $(F \wedge G)^c = F^c \vee G^c$
- $(F \vee G)^c = F^c \wedge G^c$
- $(\langle a \rangle F)^c = [a]F^c$
- $([a]F)^c = \langle a \rangle F^c$

Theorem (F^c is equivalent to the negation of F)

For any $p \in Proc$ and any HML formula F

1. $p \models F \Longrightarrow p \not\models F^c$
2. $p \not\models F \Longrightarrow p \models F^c$

Checking Validity of HML Formulae

1. Decompose the HML formula into all its subformulas
2. Starting with the smallest subformula, label all states of the LTS where it holds
3. Repeat the previous step for the smallest remaining formula
4. If the state is labeled with the formula to be checked the formula is valid that state, otherwise, it is invalid.

Checking Validity of HML Formulae

1. Decompose the HML formula into all its subformulas
2. Starting with the smallest subformula, label all states of the LTS where it holds
3. Repeat the previous step for the smallest remaining formula
4. If the state is labeled with the formula to be checked the formula is valid that state, otherwise, it is invalid.

Checking Validity of HML Formulae

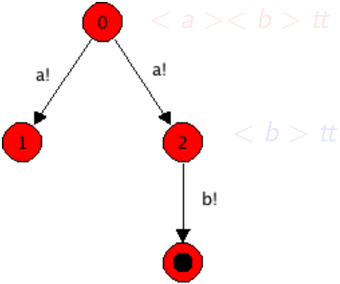
1. Decompose the HML formula into all its subformulas
2. Starting with the smallest subformula, label all states of the LTS where it holds
3. Repeat the previous step for the smallest remaining formula
4. If the state is labeled with the formula to be checked the formula is valid that state, otherwise, it is invalid.

Checking Validity of HML Formulae

1. Decompose the HML formula into all its subformulas
2. Starting with the smallest subformula, label all states of the LTS where it holds
3. Repeat the previous step for the smallest remaining formula
4. If the state is labeled with the formula to be checked the formula is valid that state, otherwise, it is invalid.

Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy the formula $< a >< b > tt$

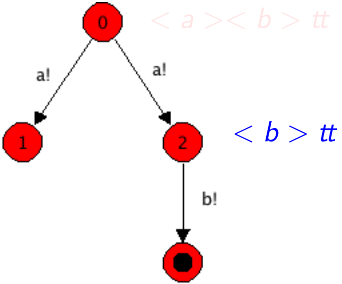


Subformulae of $< a >< b > tt$:

$$tt \qquad < b > tt \qquad < a >< b > tt$$

Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy the formula $< a >< b > tt$

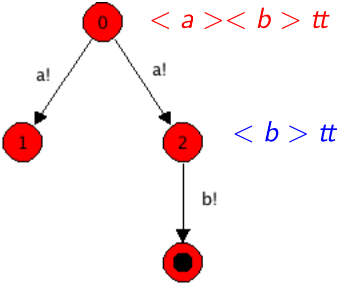


Subformulae of $< a >< b > tt$:

$tt \qquad < b > tt \qquad < a >< b > tt$

Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy the formula $< a >< b > tt$

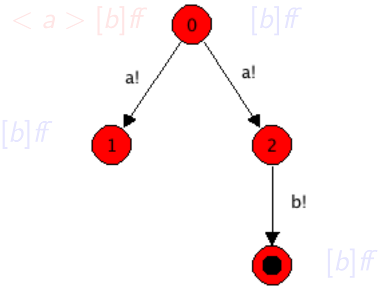


Subformulae of $< a >< b > tt$:

$tt \qquad < b > tt \qquad < a >< b > tt$

Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy formula $< a > [b]ff$

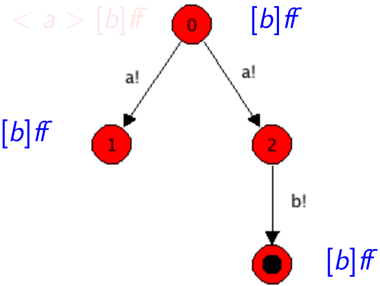


Subformulae of $< a > [b]ff$:

$ff \qquad [b]ff \qquad < a > [b]ff$

Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy formula $< a > [b]ff$

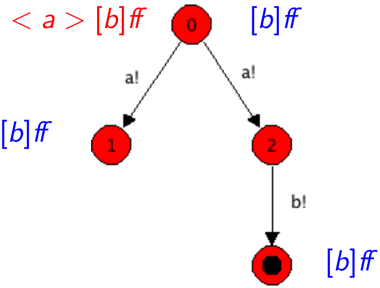


Subformulae of $< a > [b]ff$:

$ff \qquad [b]ff \qquad < a > [b]ff$

Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy formula $< a > [b]ff$

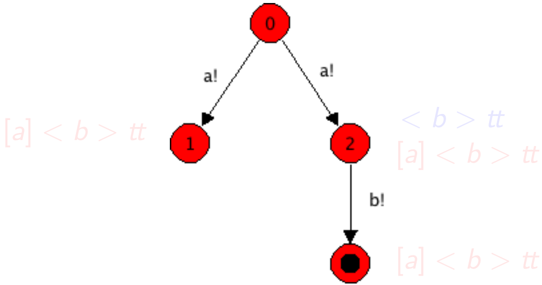


Subformulae of $< a > [b]ff$:

$ff \qquad [b]ff \qquad < a > [b]ff$

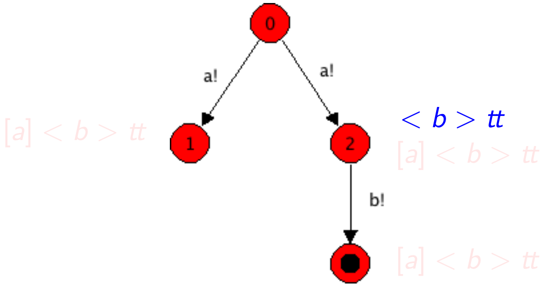
Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy formula $[a] < b > tt$



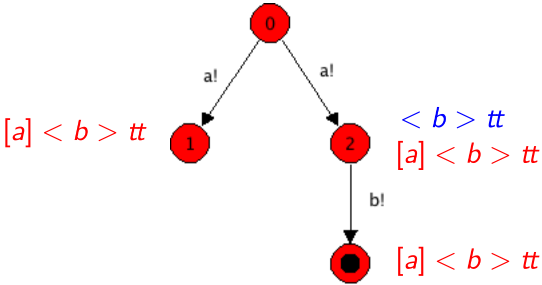
Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy formula $[a] < b > tt$



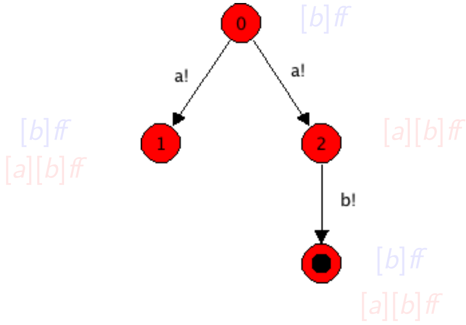
Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy formula $[a] < b > tt$



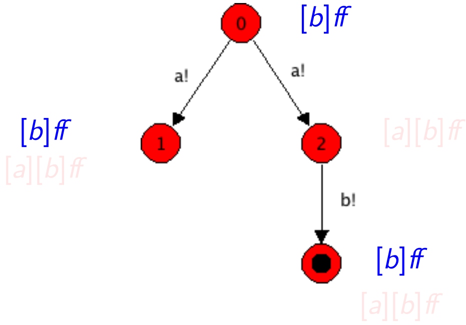
Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy formula $[a][b]ff$



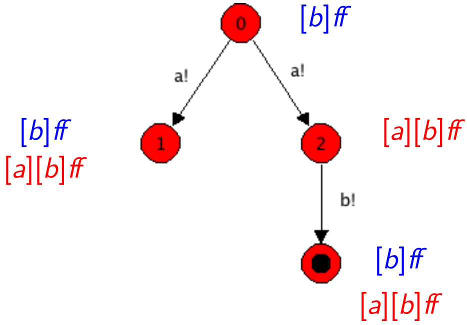
Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy formula $[a][b]ff$



Examples of Model Checking

Does the transition system corresponding to $a.nil + a.b.nil$ satisfy formula $[a][b]ff$



HML and Bisimulation

Examples

- $a.(b.nil + c.nil) \models \langle a\rangle(\langle b\rangle tt \wedge \langle c\rangle tt)$
- $a.b.nil + a.c.nil \not\models \langle a\rangle(\langle b\rangle tt \wedge \langle c\rangle tt)$

- $a.b.nil \models [a]\langle b\rangle tt$
- $a.b.nil + a.nil \not\models [a]\langle b\rangle tt$

- $a.b.(c.nil + d.nil) \models [a]\langle b\rangle\langle c\rangle tt$
- $a.b.c.nil + a.b.d.nil \not\models [a]\langle b\rangle\langle c\rangle tt$

- $a.(b.c.nil + b.d.nil) \models [a](\langle b\rangle\langle c\rangle tt \wedge \langle b\rangle\langle d\rangle tt)$
- $a.b.c.nil + a.b.d.nil \not\models [a](\langle b\rangle\langle c\rangle tt \wedge \langle b\rangle\langle d\rangle tt)$

HML and Bisimulation

Examples

- $a.(b.nil + c.nil) \models \langle a \rangle(\langle b \rangle tt \wedge \langle c \rangle tt)$
- $a.b.nil + a.c.nil \not\models \langle a \rangle(\langle b \rangle tt \wedge \langle c \rangle tt)$

- $a.b.nil \models [a]\langle b \rangle tt$
- $a.b.nil + a.nil \not\models [a]\langle b \rangle tt$

- $a.b.(c.nil + d.nil) \models [a]\langle b \rangle\langle c \rangle tt$
- $a.b.c.nil + a.b.d.nil \not\models [a]\langle b \rangle\langle c \rangle tt$

- $a.(b.c.nil + b.d.nil) \models [a](\langle b \rangle\langle c \rangle tt \wedge \langle b \rangle\langle d \rangle tt)$
- $a.b.c.nil + a.b.d.nil \not\models [a](\langle b \rangle\langle c \rangle tt \wedge \langle b \rangle\langle d \rangle tt)$

HML and Bisimulation

Examples

- $a.(b.nil + c.nil) \models \langle a \rangle(\langle b \rangle tt \wedge \langle c \rangle tt)$
- $a.b.nil + a.c.nil \not\models \langle a \rangle(\langle b \rangle tt \wedge \langle c \rangle tt)$

- $a.b.nil \models [a]\langle b \rangle tt$
- $a.b.nil + a.nil \not\models [a]\langle b \rangle tt$

- $a.b.(c.nil + d.nil) \models [a]\langle b \rangle\langle c \rangle tt$
- $a.b.c.nil + a.b.d.nil \not\models [a]\langle b \rangle\langle c \rangle tt$

- $a.(b.c.nil + b.d.nil) \models [a](\langle b \rangle\langle c \rangle tt \wedge \langle b \rangle\langle d \rangle tt)$
- $a.b.c.nil + a.b.d.nil \not\models [a](\langle b \rangle\langle c \rangle tt \wedge \langle b \rangle\langle d \rangle tt)$

HML and Bisimulation

Examples

- $a.(b.nil + c.nil) \models \langle a \rangle (\langle b \rangle tt \wedge \langle c \rangle tt)$
- $a.b.nil + a.c.nil \not\models \langle a \rangle (\langle b \rangle tt \wedge \langle c \rangle tt)$

- $a.b.nil \models [a]\langle b \rangle tt$
- $a.b.nil + a.nil \not\models [a]\langle b \rangle tt$

- $a.b.(c.nil + d.nil) \models [a]\langle b \rangle \langle c \rangle tt$
- $a.b.c.nil + a.b.d.nil \not\models [a]\langle b \rangle \langle c \rangle tt$

- $a.(b.c.nil + b.d.nil) \models [a](\langle b \rangle \langle c \rangle tt \wedge \langle b \rangle \langle d \rangle tt)$
- $a.b.c.nil + a.b.d.nil \not\models [a](\langle b \rangle \langle c \rangle tt \wedge \langle b \rangle \langle d \rangle tt)$

HML and Bisimulation

Theorem

$P \sim Q$ if and only if $P \models F \Leftrightarrow Q \models F$ for every HML formula F.

Proof

($\Longrightarrow$) Proceeds by induction on F. The interesting case is $[a]F$.
($\Longleftarrow$) We show that the set $\mathcal{S}$ of all pair of processes that satisfy the same HML formulae is a bisimulation. Suppose $\mathcal{S}$ is not a bisimulation. Then, there exists a pair $< P, Q > \in \mathcal{S}$ such that Q cannot match a move $P \xrightarrow{a} P'$. There are two cases.
Case 1: Q does not have a transition $Q \xrightarrow{a} Q'$, but then clearly P and Q do not satisfy the same formulae.
Case 2: for every evolution of $Q \xrightarrow{a} Q'$, Q' and P' do not satisfy the same formulae. Then, it is possible to construct a formula (of the form $\langle a \rangle F$ with $F = F_1 \wedge \ldots \wedge F_n$) that P satisfies but Q does not.

HML and Bisimulation: a Remark

Remark

The ($\Longrightarrow$) implication of the theorem holds for arbitrary processes.
The ($\Longleftarrow$) implication of the theorem holds for **image-finite** processes only, but not in general. This is because the construction of the formula $\langle a \rangle F$ with $F = F_1 \wedge \ldots \wedge F_n$ in the ($\Longleftarrow$)-part of the theorem is possible only when Q is image-finite.

Definition

A process P is **image-finite** if for any action a the set

$$\{ P' \mid P \xrightarrow{a} P' \}$$

is finite.

Is Hennessy-Milner Logic Powerful Enough?

Idea: a formula F can "see" only upto its depth - $md(F)$

Modal depth (nesting degree) for Hennessy-Milner formulae:

- $md(tt) = md(ff) = 0$
- $md(F \wedge G) = md(F \vee G) = \max\{md(F), md(G)\}$
- $md([a]F) = md(\langle a \rangle F) = md(F) + 1$

Theorem

Let F be a HML formula and $k = md(F)$. If the defender has a defending strategy in the strong bisimulation game between s and t up to k rounds then $s \models F$ if and only if $t \models F$.

Conclusion

There is no HML formula F that can detect a deadlock in an arbitrary LTS: deadlock might happen after a trace of length greater than $md(F)$.

Is Hennessy-Milner Logic Powerful Enough?

Idea: a formula F can "see" only upto its depth - $md(F)$

Modal depth (nesting degree) for Hennessy-Milner formulae:

- $md(tt) = md(ff) = 0$
- $md(F \wedge G) = md(F \vee G) = \max\{md(F), md(G)\}$
- $md([a]F) = md(\langle a \rangle F) = md(F) + 1$

Theorem

Let F be a HML formula and $k = md(F)$. If the defender has a defending strategy in the strong bisimulation game between s and t up to k rounds then $s \models F$ if and only if $t \models F$.

Conclusion

There is no HML formula F that can detect a deadlock in an arbitrary LTS: deadlock might happen after a trace of length greater than $md(F)$.

Is Hennessy-Milner Logic Powerful Enough?

Idea: a formula F can “see” only upto its depth - $md(F)$

Modal depth (nesting degree) for Hennessy-Milner formulae:

- $md(tt) = md(ff) = 0$
- $md(F \wedge G) = md(F \vee G) = \max\{md(F), md(G)\}$
- $md([a]F) = md(\langle a \rangle F) = md(F) + 1$

Theorem

Let F be a HML formula and $k = md(F)$. If the defender has a defending strategy in the strong bisimulation game between s and t up to k rounds then $s \models F$ if and only if $t \models F$.

Conclusion

There is no HML formula F that can detect a deadlock in an arbitrary LTS: deadlock might happen after a trace of length greater than $md(F)$.

Temporal Properties not Expressible in HM Logic

Two basic temporal properties

Can properties *Inev*(F) and *Poss*(F) where

- $s \models Inev(F)$ iff all states reachable from s satisfy F
- $s \models Poss(F)$ iff there exists a reachable state which satisfies F

be expressed as HML formulae?

Idea: Use infinite conjunction and disjunction

Let $Act = \{a_1, a_2, \ldots, a_n\}$ be a finite set of actions. We define

- $\langle Act \rangle F \stackrel{\text{def}}{=} \langle a_1 \rangle F \vee \langle a_2 \rangle F \vee \ldots \vee \langle a_n \rangle F$
- $[Act]F \stackrel{\text{def}}{=} [a_1]F \wedge [a_2]F \wedge \ldots \wedge [a_n]F$

then we can define:

- $Inev(F) \equiv F \wedge [Act]F \wedge [Act][Act]F \wedge [Act][Act][Act]F \wedge \ldots$
- $Poss(F) \equiv F \vee \langle Act \rangle F \vee \langle Act \rangle \langle Act \rangle F \vee \langle Act \rangle \langle Act \rangle \langle Act \rangle F \vee \ldots$

Temporal Properties not Expressible in HM Logic

Two basic temporal properties

Can properties $Inev(F)$ and $Poss(F)$ where

- $s \models Inev(F)$ iff all states reachable from s satisfy F
- $s \models Poss(F)$ iff there exists a reachable state which satisfies F

be expressed as HML formulae?

Idea: Use infinite conjunction and disjunction

Let $Act = \{a_1, a_2, \ldots, a_n\}$ be a finite set of actions. We define

- $\langle Act\rangle F \stackrel{\text{def}}{=} \langle a_1\rangle F \vee \langle a_2\rangle F \vee \ldots \vee \langle a_n\rangle F$
- $[Act]F \stackrel{\text{def}}{=} [a_1]F \wedge [a_2]F \wedge \ldots \wedge [a_n]F$

then we can define:

- $Inev(F) \equiv F \wedge [Act]F \wedge [Act][Act]F \wedge [Act][Act][Act]F \wedge \ldots$
- $Poss(F) \equiv F \vee \langle Act\rangle F \vee \langle Act\rangle\langle Act\rangle F \vee \langle Act\rangle\langle Act\rangle\langle Act\rangle F \vee \ldots$

Infinite Conjunctions and Disjunctions vs. Recursion

Problems

- Infinite formulae are not allowed in HML
- Infinite formulae are difficult to handle

Solution: Use recursion!

- $Inev(F)$ can be expressed by $X \stackrel{\text{def}}{=} F \wedge [Act]X$
- $Poss(F)$ can expressed by $X \stackrel{\text{def}}{=} F \vee \langle Act \rangle X$

However, to do that, we need to provide appropriate syntax and semantics.

Infinite Conjunctions and Disjunctions vs. Recursion

Problems

- Infinite formulae are not allowed in HML
- Infinite formulae are difficult to handle

Solution: Use recursion!

- *Inev*(F) can be expressed by $X \stackrel{\text{def}}{=} F \wedge [Act]X$
- *Poss*(F) can expressed by $X \stackrel{\text{def}}{=} F \vee \langle Act\rangle X$

However, to do that, we need to provide appropriate syntax and semantics.

Infinite Conjunctions and Disjunctions vs. Recursion

Problems

- Infinite formulae are not allowed in HML
- Infinite formulae are difficult to handle

Solution: Use recursion!

- *Inev*(F) can be expressed by $X \stackrel{\text{def}}{=} F \wedge [Act]X$
- *Poss*(F) can expressed by $X \stackrel{\text{def}}{=} F \vee \langle Act \rangle X$

However, to do that, we need to provide appropriate syntax and semantics.

Infinite Conjunctions and Disjunctions vs. Recursion

Syntax of Formulae

$$F ::= X \mid tt \mid ff \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle a \rangle F \mid [a]F$$

where

- $a \in Act$
- X is a variable definition:

  $X \overset{\min}{=} F_X$ or $X \overset{\max}{=} F_X$

  and F_X is a formula of the logic that can contain X.

Question:

How to define the semantics of $X \overset{\min}{=} F_X$ and $X \overset{\max}{=} F_X$?

Answer:

Use Fixed Points to assign a meaning to recursive definitions!

Infinite Conjunctions and Disjunctions vs. Recursion

Syntax of Formulae

$$F ::= X \mid tt \mid ff \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle a \rangle F \mid [a]F$$

where

- $a \in Act$
- X is a variable definition:

  $X \stackrel{\min}{=} F_X$ or $X \stackrel{\max}{=} F_X$

  and F_X is a formula of the logic that can contain X.

Question:

How to define the semantics of $X \stackrel{\min}{=} F_X$ and $X \stackrel{\max}{=} F_X$?

Answer:

Use Fixed Points to assign a meaning to recursive definitions!

Infinite Conjunctions and Disjunctions vs. Recursion

Syntax of Formulae

$$F ::= X \mid tt \mid ff \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle a \rangle F \mid [a]F$$

where

- $a \in Act$
- X is a variable definition:

  $X \overset{\min}{=} F_X$ or $X \overset{\max}{=} F_X$

  and F_X is a formula of the logic that can contain X.

Question:

How to define the semantics of $X \overset{\min}{=} F_X$ and $X \overset{\max}{=} F_X$?

Answer:

Use Fixed Points to assign a meaning to recursive definitions!

Solving Recursive Equations is Tricky

Equations over Natural Numbers ($n \in \mathbb{N}$)

$n = 2 * n$	one solution $n = 0$
$n = n + 1$	no solution
$n = 1 * n$	many solutions (every $n \in Nat$ is a solution)

Equations over Sets of Integers ($M \in 2^{\mathbb{N}}$)

$M = \{7\} \cap M$	two solutions $M = \{7\}$ and $M = \emptyset$
$M = \mathbb{N} \setminus M$	no solution
$M = \{3\} \cup M$	many solutions (every $M \supseteq \{3\}$ is a solution)

What about Equations over Processes?

To solve $X \stackrel{\text{def}}{=} [a]ff \vee \langle a \rangle X$ we need to find a set of processes $S \subseteq 2^{Proc}$ such that $S = [\cdot a \cdot]\emptyset \cup \langle \cdot a \cdot \rangle S$

Solving Recursive Equations is Tricky

Equations over Natural Numbers ($n \in \mathbb{N}$)

$n = 2 * n$	one solution $n = 0$
$n = n + 1$	no solution
$n = 1 * n$	many solutions (every $n \in Nat$ is a solution)

Equations over Sets of Integers ($M \in 2^{\mathbb{N}}$)

$M = \{7\} \cap M$	two solutions $M = \{7\}$ and $M = \emptyset$
$M = \mathbb{N} \setminus M$	no solution
$M = \{3\} \cup M$	many solutions (every $M \supseteq \{3\}$ is a solution)

What about Equations over Processes?

To solve $X \stackrel{\text{def}}{=} [a]ff \vee \langle a \rangle X$ we need to find a set of processes $S \subseteq 2^{Proc}$ such that $S = [\cdot a \cdot]\emptyset \cup \langle \cdot a \cdot \rangle S$

Solving Recursive Equations is Tricky

Equations over Natural Numbers ($n \in \mathbb{N}$)

$n = 2 * n$ one solution $n = 0$
$n = n + 1$ no solution
$n = 1 * n$ many solutions (every $n \in Nat$ is a solution)

Equations over Sets of Integers ($M \in 2^{\mathbb{N}}$)

$M = \{7\} \cap M$ two solutions $M = \{7\}$ and $M = \emptyset$
$M = \mathbb{N} \setminus M$ no solution
$M = \{3\} \cup M$ many solutions (every $M \supseteq \{3\}$ is a solution)

What about Equations over Processes?

To solve $X \stackrel{\text{def}}{=} [a]\mathit{ff} \vee \langle a \rangle X$ we need to find a set of processes $S \subseteq 2^{Proc}$ such that $S = [\cdot a \cdot]\emptyset \cup \langle \cdot a \cdot \rangle S$

Denotational Semantics for HML - without recursion

Idea: $[\![F]\!]$ is the set of all states that satisfy F

- $[\![tt]\!] = Proc$
- $[\![ff]\!] = \emptyset$
- $[\![F \vee G]\!] = [\![F]\!] \cup [\![G]\!]$
- $[\![F \wedge G]\!] = [\![F]\!] \cap [\![G]\!]$
- $[\![\langle a \rangle F]\!] = \langle \cdot a \cdot \rangle [\![F]\!]$
- $[\![[a] F]\!] = [\cdot a \cdot] [\![F]\!]$

where $\langle \cdot a \cdot \rangle, [\cdot a \cdot] : 2^{(Proc)} \to 2^{(Proc)}$ are defined by:

- $\langle \cdot a \cdot \rangle S = \{p \in Proc \mid \exists p'.\ p \xrightarrow{a} p' \text{ and } p' \in S\}$
- $[\cdot a \cdot] S = \{p \in Proc \mid \forall p'.\ p \xrightarrow{a} p' \implies p' \in S\}$.

Denotational Semantics for HML - without recursion

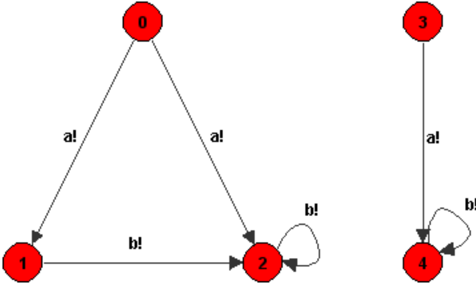
Idea: $[\![F]\!]$ is the set of all states that satisfy F

- $[\![tt]\!] = Proc$
- $[\![ff]\!] = \emptyset$
- $[\![F \vee G]\!] = [\![F]\!] \cup [\![G]\!]$
- $[\![F \wedge G]\!] = [\![F]\!] \cap [\![G]\!]$
- $[\![\langle a \rangle F]\!] = \langle \cdot a \cdot \rangle [\![F]\!]$
- $[\![[a]F]\!] = [\cdot a \cdot][\![F]\!]$

where $\langle \cdot a \cdot \rangle, [\cdot a \cdot] : 2^{(Proc)} \to 2^{(Proc)}$ are defined by:

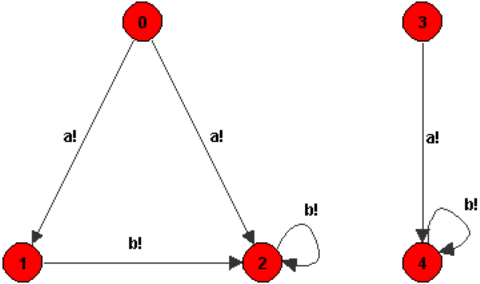
- $\langle \cdot a \cdot \rangle S = \{p \in Proc \mid \exists p'.\ p \xrightarrow{a} p' \text{ and } p' \in S\}$
- $[\cdot a \cdot]S = \{p \in Proc \mid \forall p'.\ p \xrightarrow{a} p' \implies p' \in S\}$.

Examples for $\langle \cdot a \cdot \rangle$ and $[\cdot a \cdot]$



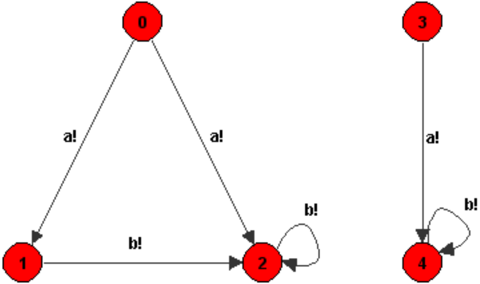
- $\langle \cdot a \cdot \rangle\{1,4\} = \{0,3\}$
- $[\cdot a \cdot]\{1,4\} = \{1,2,3,4\}$

Examples for $\langle \cdot a \cdot \rangle$ and $[\cdot a \cdot]$



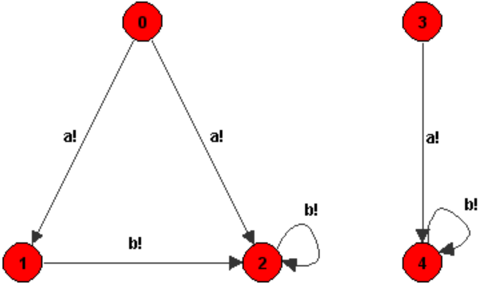
- $\langle \cdot a \cdot \rangle \{1, 4\} = \{0, 3\}$
- $[\cdot a \cdot]\{1, 4\} = \{1, 2, 3, 4\}$

Examples for $\langle \cdot a \cdot \rangle$ and $[\cdot a \cdot]$



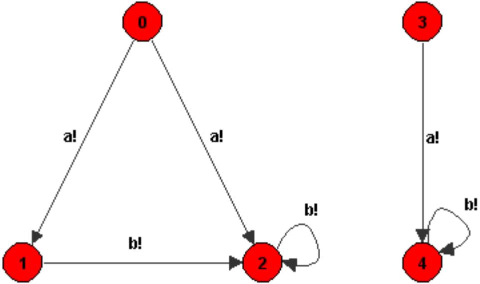
- $\langle \cdot a \cdot \rangle \{1, 4\} = \{0, 3\}$
- $[\cdot a \cdot] \{1, 4\} = \{1, 2, 3, 4\}$

Examples for $\langle \cdot a \cdot \rangle$ and $[\cdot a \cdot]$



- $\langle \cdot a \cdot \rangle \{1, 4\} = \{0, 3\}$
- $[\cdot a \cdot] \{1, 4\} = \{1, 2, 3, 4\}$

Examples for $\langle \cdot a \cdot \rangle$ and $[\cdot a \cdot]$



- $\langle \cdot a \cdot \rangle \{1, 4\} = \{0, 3\}$
- $[\cdot a \cdot]\{1, 4\} = \{1, 2, 3, 4\}$

The Correspondence Theorem

Theorem

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ *be an LTS,* $p \in Proc$ *and* F *a formula of Hennessy-Milner logic. Then*

$$p \models F \quad \textit{if and only if} \quad p \in [\![F]\!].$$

Proof: by structural induction on the structure of the formula F.

The Correspondence Theorem

Theorem

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS, $p \in Proc$ and F a formula of Hennessy-Milner logic. Then

$$p \models F \quad \textit{if and only if} \quad p \in [\![F]\!].$$

Proof: by structural induction on the structure of the formula F.

Image-Finite Labelled Transition System

Image-Finite System

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS. We call it image-finite iff for every $p \in Proc$ and every $a \in Act$ the set

$$\{p' \in Proc \mid p \xrightarrow{a} p'\}$$

is finite.

Relationship between HML and Strong Bisimilarity

Theorem (Hennessy-Milner)

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an image-finite LTS and $p, q \in St$. Then

$$p \sim q$$

if and only if

for every HML formula F: $(p \models F \iff q \models F)$.

Denotational Semantics for HML with recursion

Syntax of HML

$$F ::= X \mid tt \mid ff \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle a \rangle F \mid [a]F$$

where $a \in Act$ and X is a distinguished variable with a definition

- $X \stackrel{\min}{=} F_X$, or $X \stackrel{\max}{=} F_X$

such that F_X is a formula of the logic that can contain X.

How to Define Semantics?

To deal with recursive variables, assumptions on the states satisfied by them are made, and for every formula F a function $O_F : 2^{Proc} \rightarrow 2^{Proc}$ is defined such that:

- if S is the set of processes that satisfy X then $O_F(S)$ is the set of processes that satisfy F.

Definition of $O_F : 2^{Proc} \rightarrow 2^{Proc}$ with $S \subseteq 2^{Proc}$

Semantics of HML Formulae with Variables

$$
\begin{aligned}
O_X(S) &= S \\
O_{tt}(S) &= Proc \\
O_{ff}(S) &= \emptyset \\
O_{F_1 \wedge F_2}(S) &= O_{F_1}(S) \cap O_{F_2}(S) \\
O_{F_1 \vee F_2}(S) &= O_{F_1}(S) \cup O_{F_2}(S) \\
O_{\langle a \rangle F}(S) &= \langle \cdot a \cdot \rangle O_F(S) \\
O_{[a] F}(S) &= [\cdot a \cdot] O_F(S)
\end{aligned}
$$

We can now deal with $X \stackrel{\text{def}}{=} F \wedge [Act]X$ and $X \stackrel{\text{def}}{=} F \vee \langle Act \rangle X$ by considering the recursive equations over set of processes:

- $O_X(S) = O_{F \wedge [Act]X}(S)$
- $O_X(S) = O_{F \vee \langle Act \rangle X}(S)$.

Definition of $O_F : 2^{Proc} \rightarrow 2^{Proc}$ with $S \subseteq 2^{Proc}$

Semantics of HML Formulae with Variables

$$\begin{array}{rcl} O_X(S) & = & S \\ O_{tt}(S) & = & Proc \\ O_{ff}(S) & = & \emptyset \\ O_{F_1 \wedge F_2}(S) & = & O_{F_1}(S) \cap O_{F_2}(S) \\ O_{F_1 \vee F_2}(S) & = & O_{F_1}(S) \cup O_{F_2}(S) \\ O_{\langle a \rangle F}(S) & = & \langle \cdot a \cdot \rangle O_F(S) \\ O_{[a] F}(S) & = & [\cdot a \cdot] O_F(S) \end{array}$$

We can now deal with $X \stackrel{\mathrm{def}}{=} F \wedge [Act]X$ and $X \stackrel{\mathrm{def}}{=} F \vee \langle Act \rangle X$ by considering the recursive equations over set of processes:

- $O_X(S) = O_{F \wedge [Act] X}(S)$
- $O_X(S) = O_{F \vee \langle Act \rangle X}(S)$.

Definition of $O_F : 2^{Proc} \rightarrow 2^{Proc}$ with $S \subseteq 2^{Proc}$

Semantics of HML Formulae with Variables

$$
\begin{array}{rcl}
O_X(S) & = & S \\
O_{tt}(S) & = & Proc \\
O_{ff}(S) & = & \emptyset \\
O_{F_1 \wedge F_2}(S) & = & O_{F_1}(S) \cap O_{F_2}(S) \\
O_{F_1 \vee F_2}(S) & = & O_{F_1}(S) \cup O_{F_2}(S) \\
O_{\langle a \rangle F}(S) & = & \langle \cdot a \cdot \rangle O_F(S) \\
O_{[a] F}(S) & = & [\cdot a \cdot] O_F(S)
\end{array}
$$

We can now deal with $X \stackrel{\text{def}}{=} F \wedge [Act]X$ and $X \stackrel{\text{def}}{=} F \vee \langle Act \rangle X$ by considering the recursive equations over set of processes:

- $O_X(S) = O_{F \wedge [Act]X}(S)$
- $O_X(S) = O_{F \vee \langle Act \rangle X}(S)$.

Alternative semantics for HML

To do that we need to find the appropriate mathematical tools for finding the (unique?) solutions for such recursive equations and look for fixed points.

The intuition behind the formal semantics of HML formulae is that each formula determines a set of states for which the formula is valid. We have however to consider that:

- This kind of equations do not necessarily determine a set of states uniquely; e.g., for formula X with $X = X$ there is no such unique set of states; any set of states is a solution of the equation.

Thus it is needed to indicate which solution is meant, e.g., whether one wants the least or the greatest solution and care needs to be taken that these solutions do exists.

General Approach – Lattice Theory

Problem

For a set D and a function $f : D \to D$, for which elements $x \in D$ we have

$$x = f(x) \ ?$$

Such x's are called fixed points.

Partially Ordered Set

Partially ordered set (or simply a partial order) is a pair $(D, \sqsubseteq)$ s.t.

- D is a set
- $\sqsubseteq \subseteq D \times D$ is a binary relation on D which is
  - reflexive: $\forall d \in D.\ d \sqsubseteq d$
  - antisymmetric: $\forall d, e \in D.\ d \sqsubseteq e \ \wedge \ e \sqsubseteq d \ \Rightarrow \ d = e$
  - transitive: $\forall d, e, f \in D.\ d \sqsubseteq e \ \wedge \ e \sqsubseteq f \ \Rightarrow \ d \sqsubseteq f$

General Approach – Lattice Theory

Problem

For a set D and a function $f : D \to D$, for which elements $x \in D$ we have

$$x = f(x) \ ?$$

Such x's are called fixed points.

Partially Ordered Set

Partially ordered set (or simply a partial order) is a pair $(D, \sqsubseteq)$ s.t.

- D is a set
- $\sqsubseteq \subseteq D \times D$ is a binary relation on D which is
  - reflexive: $\forall d \in D.\ d \sqsubseteq d$
  - antisymmetric: $\forall d, e \in D.\ d \sqsubseteq e \ \wedge \ e \sqsubseteq d \ \Rightarrow \ d = e$
  - transitive: $\forall d, e, f \in D.\ d \sqsubseteq e \ \wedge \ e \sqsubseteq f \ \Rightarrow \ d \sqsubseteq f$

Supremum and Infimum

Upper/Lower Bounds (Let $X \subseteq D$)

- $d \in D$ is an upper bound for X (written $X \sqsubseteq d$) iff $x \sqsubseteq d$ for all $x \in X$
- $d \in D$ is a lower bound for X (written $d \sqsubseteq X$) iff $d \sqsubseteq x$ for all $x \in X$

Least Upper Bound and Greatest Lower Bound (Let $X \subseteq D$)

- $d \in D$ is the least upper bound (supremum) for X ($\sqcup X$) iff
  1. $X \sqsubseteq d$
  2. $\forall d' \in D.\ X \sqsubseteq d' \Rightarrow d \sqsubseteq d'$
- $d \in D$ is the greatest lower bound (infimum) for X ($\sqcap X$) iff
  1. $d \sqsubseteq X$
  2. $\forall d' \in D.\ d' \sqsubseteq X \Rightarrow d' \sqsubseteq d$

Supremum and Infimum

Upper/Lower Bounds (Let $X \subseteq D$)

- $d \in D$ is an upper bound for X (written $X \sqsubseteq d$) iff $x \sqsubseteq d$ for all $x \in X$
- $d \in D$ is a lower bound for X (written $d \sqsubseteq X$) iff $d \sqsubseteq x$ for all $x \in X$

Least Upper Bound and Greatest Lower Bound (Let $X \subseteq D$)

- $d \in D$ is the least upper bound (supremum) for X ($\sqcup X$) iff
  1. $X \sqsubseteq d$
  2. $\forall d' \in D.\ X \sqsubseteq d' \ \Rightarrow\ d \sqsubseteq d'$
- $d \in D$ is the greatest lower bound (infimum) for X ($\sqcap X$) iff
  1. $d \sqsubseteq X$
  2. $\forall d' \in D.\ d' \sqsubseteq X \ \Rightarrow\ d' \sqsubseteq d$

Complete Lattices and Monotonic Functions

Complete Lattice

A partially ordered set $(D, \sqsubseteq)$ is called complete lattice iff $\sqcup X$ and $\sqcap X$ exist for any $X \subseteq D$.

We define the top and bottom by $\top \stackrel{\text{def}}{=} \sqcup D$ and $\bot \stackrel{\text{def}}{=} \sqcap D$.

Monotonic Function and Fixed Points

A function $f : D \to D$ is called monotonic iff

$$d \sqsubseteq e \quad \Rightarrow \quad f(d) \sqsubseteq f(e)$$

for all $d, e \in D$.

Element $d \in D$ is called fixed point iff $d = f(d)$.

Complete Lattices and Monotonic Functions

Complete Lattice

A partially ordered set $(D, \sqsubseteq)$ is called complete lattice iff $\sqcup X$ and $\sqcap X$ exist for any $X \subseteq D$.

We define the top and bottom by $\top \stackrel{\text{def}}{=} \sqcup D$ and $\bot \stackrel{\text{def}}{=} \sqcap D$.

Monotonic Function and Fixed Points

A function $f : D \to D$ is called monotonic iff

$$d \sqsubseteq e \quad \Rightarrow \quad f(d) \sqsubseteq f(e)$$

for all $d, e \in D$.

Element $d \in D$ is called fixed point iff $d = f(d)$.

Tarski's Fixed Point Theorem

Theorem (Tarski)

Let $(D, \sqsubseteq)$ be a complete lattice and let $f : D \to D$ be a monotonic function.

Then f has a unique largest fixed point z_{max} and a unique least fixed point z_{min} given by:

$$z_{max} \stackrel{\text{def}}{=} \bigsqcup\{x \in D \mid x \sqsubseteq f(x)\}$$

$$z_{min} \stackrel{\text{def}}{=} \bigsqcap\{x \in D \mid f(x) \sqsubseteq x\}$$

Computing Min and Max Fixed Points on Finite Lattices

Let $(D, \sqsubseteq)$ be a complete lattice and $f : D \to D$ monotonic.

Let $f^1(x) \stackrel{\text{def}}{=} f(x)$ and $f^n(x) \stackrel{\text{def}}{=} f(f^{n-1}(x))$ for $n > 1$, i.e.,

$$f^n(x) = \underbrace{f(f(\dots f}_{n \text{ times}}(x)\dots)).$$

Theorem

If D is a finite set then there exist integers $M, m > 0$ such that

- $z_{max} = f^M(\top)$
- $z_{min} = f^m(\bot)$

Idea (for z_{min} and z_{max})

The following sequences stabilize for any finite D

- $\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \sqsubseteq f(f(f(\bot))) \sqsubseteq \cdots$
- $D \sqsupseteq f(D) \sqsupseteq f(f(D)) \sqsupseteq f(f(f(D))) \sqsupseteq \cdots$

Computing Min and Max Fixed Points on Finite Lattices

Let $(D, \sqsubseteq)$ be a complete lattice and $f : D \to D$ monotonic.

Let $f^1(x) \stackrel{\text{def}}{=} f(x)$ and $f^n(x) \stackrel{\text{def}}{=} f(f^{n-1}(x))$ for $n > 1$, i.e.,

$$f^n(x) = \underbrace{f(f(\ldots f}_{n \text{ times}}(x)\ldots)).$$

Theorem

If D is a finite set then there exist integers $M, m > 0$ such that

- $z_{max} = f^M(\top)$
- $z_{min} = f^m(\bot)$

Idea (for z_{min} and z_{max})

The following sequences stabilize for any finite D

- $\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \sqsubseteq f(f(f(\bot))) \sqsubseteq \cdots$
- $D \sqsupseteq f(D) \sqsupseteq f(f(D)) \sqsupseteq f(f(f(D))) \sqsupseteq \cdots$

Monotonic Functions over Sets of Processes

Fixed Points of Functions Sets of Processes

A function $f : 2^{Proc} \rightarrow 2^{Proc}$ is called monotonic iff

$$X \subseteq Y \quad \Rightarrow \quad f(X) \subseteq f(Y)$$

for all $X, Y \in 2^{Proc}$.

A set $X \in 2^{Proc}$ is called a fixed point of f iff $X = f(X)$.

Questions

Is the function $f(X) = X \cup \{s, t\}$ monotonic? What about $g(X) = Proc \setminus X$? Do these functions have fixed points?

Tarski's Fixed Point Theorem for Processes

Theorem (Tarski)

Let $f : 2^{Proc} \rightarrow 2^{Proc}$ be a monotonic function.
Then f has a unique largest fixed point z_{max} and a unique least fixed point z_{min} given by:

$$z_{max} \stackrel{\text{def}}{=} \bigcup \{X \in 2^{Proc} \mid X \subseteq f(X)\}$$

$$z_{min} \stackrel{\text{def}}{=} \bigcap \{X \in 2^{Proc} \mid f(X) \subseteq X\}$$

Computing Fixed Points on Finite Sets of Processes

Let $f : 2^{Proc} \to 2^{Proc}$ be monotonic.
Let $f^1(X) \stackrel{\text{def}}{=} f(X)$ and $f^n(X) \stackrel{\text{def}}{=} f(f^{n-1}(X))$ for $n > 1$, i.e.,

$$f^n(X) = \underbrace{f(f(\ldots f}_{n \text{ times}}(X)\ldots)).$$

Theorem

If 2^{Proc} is a finite set then there exist integers $M, m > 0$ such that

- $z_{max} = f^M(Proc)$
- $z_{min} = f^m(\emptyset)$

The following sequences stabilize for any finite set *Proc* of processes

- $\emptyset \subseteq f(\emptyset) \subseteq f(f(\emptyset)) \subseteq f(f(f(\emptyset))) \subseteq \cdots$
- $Proc \supseteq f(Proc) \supseteq f(f(Proc)) \supseteq f(f(f(Proc))) \supseteq \cdots$

Computing Fixed Points on Finite Sets of Processes

Let $f : 2^{Proc} \rightarrow 2^{Proc}$ be monotonic.
Let $f^1(X) \stackrel{\text{def}}{=} f(X)$ and $f^n(X) \stackrel{\text{def}}{=} f(f^{n-1}(X))$ for $n > 1$, i.e.,

$$f^n(X) = \underbrace{f(f(\ldots f}_{n \text{ times}}(X)\ldots)).$$

Theorem

If 2^{Proc} is a finite set then there exist integers $M, m > 0$ such that

- $z_{max} = f^M(Proc)$
- $z_{min} = f^m(\emptyset)$

The following sequences stabilize for any finite set *Proc* of processes

- $\emptyset \subseteq f(\emptyset) \subseteq f(f(\emptyset)) \subseteq f(f(f(\emptyset))) \subseteq \cdots$
- $Proc \supseteq f(Proc) \supseteq f(f(Proc)) \supseteq f(f(f(Proc))) \supseteq \cdots$

Tarski's Fixed Point Theorem – Summary

Let $(D, \sqsubseteq)$ be a complete lattice and let $f : D \to D$ be a monotonic function.

Tarski's Fixed Point Theorem

Then f has a unique largest fixed point z_{max} and a unique least fixed point z_{min} given by:

$$z_{max} \stackrel{\text{def}}{=} \sqcup\{x \in D \mid x \sqsubseteq f(x)\}$$

$$z_{min} \stackrel{\text{def}}{=} \sqcap\{x \in D \mid f(x) \sqsubseteq x\}$$

Computing Fixed Points in Finite Lattices

If D is a finite set then there exist integers $M, m > 0$ such that

- $z_{max} = f^M(\top)$
- $z_{min} = f^m(\bot)$

HML with One Recursively Defined Variable

Syntax of Formulae

Formulae are given by the following abstract syntax

$$F ::= X \mid tt \mid ff \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle a \rangle F \mid [a]F$$

where $a \in Act$ and X is a distinguished variable with a definition

- $X \overset{\min}{=} F_X$, or $X \overset{\max}{=} F_X$

such that F_X is a formula of the logic (can contain X).

How to Define Semantics?

In order to deal with recursion variable X, we make assumption on the states satisfied by X and for every formula F we define a function $O_F : 2^{Proc} \to 2^{Proc}$ such that:

- if S is the set of processes that satisfy X then $O_F(S)$ is the set of processes that satisfy F.

HML with One Recursively Defined Variable

Syntax of Formulae

Formulae are given by the following abstract syntax

$$F ::= X \mid tt \mid ff \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle a \rangle F \mid [a]F$$

where $a \in Act$ and X is a distinguished variable with a definition

- $X \stackrel{\min}{=} F_X$, or $X \stackrel{\max}{=} F_X$

such that F_X is a formula of the logic (can contain X).

How to Define Semantics?

In order to deal with recursion variable X, we make assumption on the states satisfied by X and for every formula F we define a function $O_F : 2^{Proc} \rightarrow 2^{Proc}$ such that:

- if S is the set of processes that satisfy X then $O_F(S)$ is the set of processes that satisfy F.

Definition of $O_F : 2^{Proc} \rightarrow 2^{Proc}$ (let $S \subseteq 2^{Proc}$)

$$
\begin{array}{rcl}
O_X(S) & = & S \\
O_{tt}(S) & = & Proc \\
O_{ff}(S) & = & \emptyset \\
O_{F_1 \wedge F_2}(S) & = & O_{F_1}(S) \cap O_{F_2}(S) \\
O_{F_1 \vee F_2}(S) & = & O_{F_1}(S) \cup O_{F_2}(S) \\
O_{\langle a \rangle F}(S) & = & \langle \cdot a \cdot \rangle O_F(S) \\
O_{[a]F}(S) & = & [\cdot a \cdot] O_F(S)
\end{array}
$$

O_F is monotonic for every formula F

$$S_1 \subseteq S_2 \;\Rightarrow\; O_F(S_1) \subseteq O_F(S_2)$$

Proof: easy (structural induction on the structure of F).

Definition of $O_F : 2^{Proc} \rightarrow 2^{Proc}$ (let $S \subseteq 2^{Proc}$)

$$
\begin{array}{rcl}
O_X(S) & = & S \\
O_{tt}(S) & = & Proc \\
O_{ff}(S) & = & \emptyset \\
O_{F_1 \wedge F_2}(S) & = & O_{F_1}(S) \cap O_{F_2}(S) \\
O_{F_1 \vee F_2}(S) & = & O_{F_1}(S) \cup O_{F_2}(S) \\
O_{\langle a \rangle F}(S) & = & \langle \cdot a \cdot \rangle O_F(S) \\
O_{[a] F}(S) & = & [\cdot a \cdot] O_F(S)
\end{array}
$$

O_F is monotonic for every formula F

$$S_1 \subseteq S_2 \;\Rightarrow\; O_F(S_1) \subseteq O_F(S_2)$$

Proof: easy (structural induction on the structure of F).

Semantics

Observation

We know O_F is monotonic, so O_F has a unique greatest and least fixed point.

Semantics of the Variable X

- If $X \stackrel{\max}{=} F_X$ then

$$[\![X]\!] = \bigcup \{S \subseteq Proc \mid S \subseteq O_{F_X}(S)\}.$$

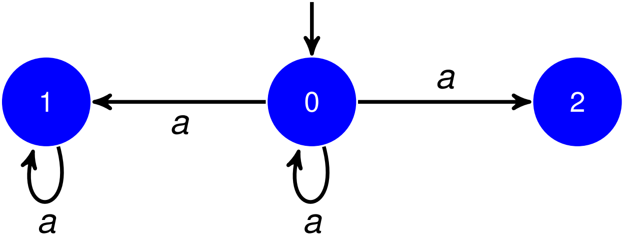
- If $X \stackrel{\min}{=} F_X$ then

$$[\![X]\!] = \bigcap \{S \subseteq Proc \mid O_{F_X}(S) \subseteq S\}.$$

Example 1

A state can be reached where *a* cannot be executed

$$X \stackrel{\text{def}}{=} [a]\mathit{false} \vee \langle Act \rangle X$$



The property is valid for the labeled transition system

Solutions of this equation are the sets: $\{0, 2\}$ and $\{0, 1, 2\}$

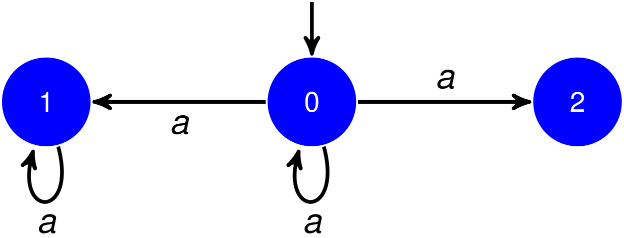
We intended to describe the least solution!

$$X \stackrel{\text{min}}{=} [a]\mathit{false} \vee \langle Act \rangle X$$

Example 1

A state can be reached where *a* cannot be executed

$$X \stackrel{\text{def}}{=} [a]\mathit{false} \vee \langle Act \rangle X$$



The property is valid for the labeled transition system

Solutions of this equation are the sets: $\{0, 2\}$ and $\{0, 1, 2\}$

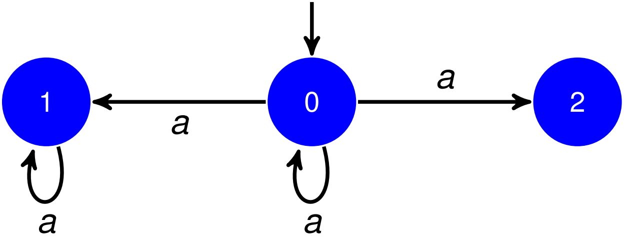
We intended to describe the least solution!

$$X \stackrel{\text{min}}{=} [a]\mathit{false} \vee \langle Act \rangle X$$

Example 1

A state can be reached where *a* cannot be executed

$$X \stackrel{\text{def}}{=} [a]\mathit{false} \vee \langle Act \rangle X$$



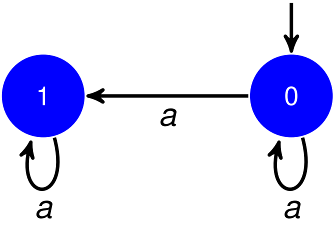
The property is valid for the labeled transition system
Solutions of this equation are the sets: $\{0, 2\}$ and $\{0, 1, 2\}$
We intended to describe the least solution!

$$X \stackrel{\text{min}}{=} [a]\mathit{false} \vee \langle Act \rangle X$$

Example 2

A state can be reached where *a* cannot be executed

$$X \stackrel{\min}{=} [a]\mathit{false} \vee \langle Act \rangle X$$



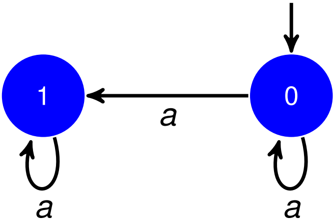
The unique least solution for this equation is the empty set of states ($\emptyset$)

Hence the property is not valid for the labeled transition system

Example 2

A state can be reached where *a* cannot be executed

$$X \stackrel{\min}{=} [a]\mathit{false} \vee \langle Act \rangle X$$

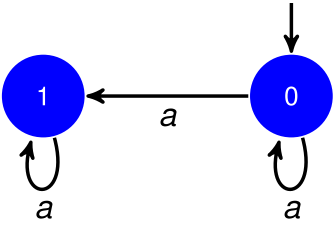


The unique least solution for this equation is the empty set of states ($\emptyset$)
Hence the property is not valid for the labeled transition system

Example 3

In every reachable state an a-transition is possible

$$X \stackrel{\text{def}}{=} [a]\mathit{false} \vee \langle Act\rangle X$$



Solutions of this equation are the sets: $\emptyset$, $\{1\}$ and $\{0, 1\}$

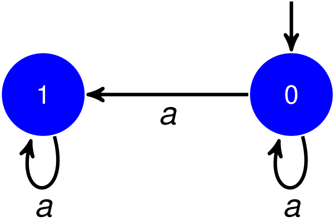
We intended to describe the greatest solution!

$$X \stackrel{\max}{=} [a]\mathit{false} \vee \langle Act\rangle X$$

Example 3

In every reachable state an a-transition is possible

$$X \stackrel{\text{def}}{=} [a]\mathit{false} \vee \langle Act \rangle X$$



Solutions of this equation are the sets: $\emptyset$, $\{1\}$ and $\{0, 1\}$

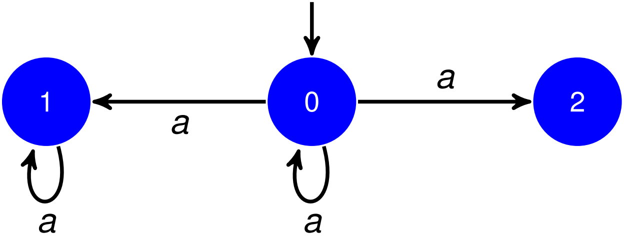
We intended to describe the greatest solution!

$$X \stackrel{\max}{=} [a]\mathit{false} \vee \langle Act \rangle X$$

Example 4

In every reachable state an a-transition is possible

$$X \stackrel{\max}{=} \langle a \rangle true \wedge [Act]X$$



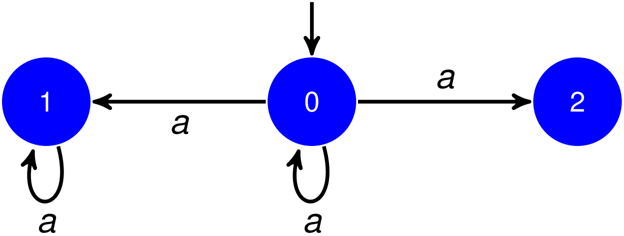
The greatest solution for this equation is the set of states: $\{1\}$

Thus property is not valid for the labeled transition system.

Example 4

In every reachable state an a-transition is possible

$$X \stackrel{\max}{=} \langle a \rangle true \wedge [Act]X$$



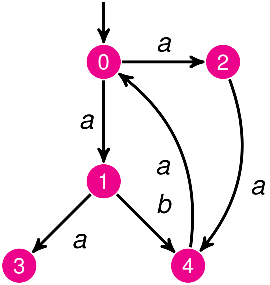
The greatest solution for this equation is the set of states: $\{1\}$

Thus property is not valid for the labeled transition system.

Example 5

There is a path of (a and b) transitions to a b-transition

$$X \stackrel{\min}{=} \langle b\rangle true \vee \langle\{a, b\}\rangle X$$



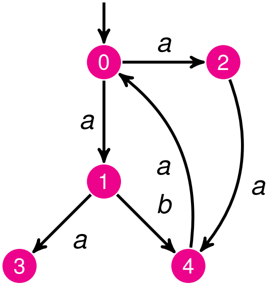
The solution for this equation is the set of states: $\{0, 1, 2, 4\}$

Thus property is valid for the labeled transition system.

Example 5

There is a path of (a and b) transitions to a b-transition

$$X \stackrel{\min}{=} \langle b \rangle true \vee \langle \{a, b\} \rangle X$$



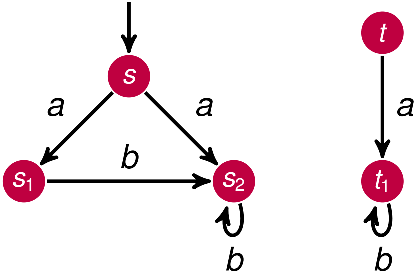
The solution for this equation is the set of states: $\{0, 1, 2, 4\}$

Thus property is valid for the labeled transition system.

Example 6

All states reachable by b-transitions (0 or more) have a b-transition

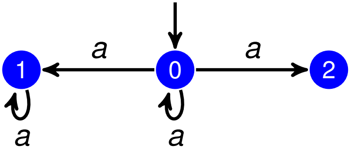
$$X \stackrel{\max}{=} \langle b\rangle true \wedge [b]X$$



The greatest solution for this equation is the set of states: $\{s_1, s_2, t_1\}$

Calculating Minimum Fixed Points

Example: $X \overset{\min}{=} [a]\mathit{false} \vee \langle Act \rangle X$

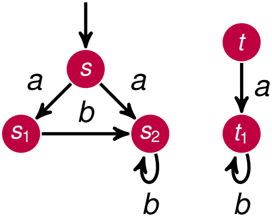


$$
\begin{aligned}
O_{F_X}(S) &= O_{[a]\mathit{false}}(S) \cup O_{\langle Act \rangle X}(S) \\
&= [\cdot a \cdot] O_{\mathit{false}}(S) \cup \langle \cdot Act \cdot \rangle O_X(S) \\
&= [\cdot a \cdot] \varnothing \cup \langle \cdot Act \cdot \rangle S \\
&= \{2\} \cup \langle \cdot Act \cdot \rangle S
\end{aligned}
$$

1. $O_{F_X}(\varnothing) = \{2\} \cup \langle \cdot Act \cdot \rangle \varnothing = \{2\} \cup \varnothing = \{2\}$
2. $O_{F_X}(\{2\}) = \{2\} \cup \langle \cdot Act \cdot \rangle \{2\} = \{2\} \cup \{0\} = \{0, 2\}$
3. $O_{F_X}(\{0, 2\}) = \{2\} \cup \langle \cdot Act \cdot \rangle \{0, 2\} = \{2\} \cup \{0\} = \{0, 2\}$

Calculating Maximum Fixed Points

Example: $X \stackrel{max}{=} \langle b \rangle true \wedge [b]X$



$$
\begin{aligned}
O_{F_X}(S) &= O_{\langle b \rangle true}(S) \cap O_{[b]X}(S) \\
&= \langle \cdot b \cdot \rangle O_{true}(S) \cap [\cdot b \cdot] O_X(S) \\
&= \langle \cdot b \cdot \rangle Proc \cap [\cdot b \cdot] S \\
&= \{s_1, s_2, t_1\} \cap [\cdot b \cdot] S
\end{aligned}
$$

1. $O_{F_X}(Proc) = \{s_1, s_2, t_1\} \cap [\cdot b \cdot] Proc = \{s_1, s_2, t_1\} \cap \{s, s_1, s_2, t, t_1\} = \{s_1, s_2, t_1\}$
2. $O_{F_X}(\{s_1, s_2, t_1\}) = \{s_1, s_2, t_1\} \cap [\cdot b \cdot]\{s_1, s_2, t_1\} =$

Selection of Temporal Properties

- *Inv*(*F*): $X \stackrel{\max}{=} F \wedge [Act]X$
- *Pos*(*F*): $X \stackrel{\min}{=} F \vee \langle Act\rangle X$

- *Safe*(*F*): $X \stackrel{\max}{=} F \wedge ([Act]\mathit{ff} \vee \langle Act\rangle X)$
- *Even*(*F*): $X \stackrel{\min}{=} F \vee (\langle Act\rangle \mathit{tt} \wedge [Act]X)$

- $F\ \mathcal{U}^w\ G$: $X \stackrel{\max}{=} G \vee (F \wedge [Act]X)$
- $F\ \mathcal{U}^s\ G$: $X \stackrel{\min}{=} G \vee (F \wedge \langle Act\rangle \mathit{tt} \wedge [Act]X)$

Using until we can express e.g. *Inv*(*F*) and *Even*(*F*):

$$Inv(F) \equiv F\ \mathcal{U}^w\ \mathit{ff} \qquad\qquad Even(F) \equiv \mathit{tt}\ \mathcal{U}^s\ F$$

Selection of Temporal Properties

- *Inv*(F): $X \stackrel{\max}{=} F \wedge [Act]X$
- *Pos*(F): $X \stackrel{\min}{=} F \vee \langle Act \rangle X$

- *Safe*(F): $X \stackrel{\max}{=} F \wedge ([Act]ff \vee \langle Act \rangle X)$
- *Even*(F): $X \stackrel{\min}{=} F \vee (\langle Act \rangle tt \wedge [Act]X)$

- $F\ \mathcal{U}^w\ G$: $X \stackrel{\max}{=} G \vee (F \wedge [Act]X)$
- $F\ \mathcal{U}^s\ G$: $X \stackrel{\min}{=} G \vee (F \wedge \langle Act \rangle tt \wedge [Act]X)$

Using until we can express e.g. *Inv*(F) and *Even*(F):

$$Inv(F) \equiv F\ \mathcal{U}^w\ ff \qquad\qquad Even(F) \equiv tt\ \mathcal{U}^s\ F$$

Selection of Temporal Properties

- $\mathit{Inv}(F)$: $X \stackrel{\max}{=} F \wedge [\mathit{Act}]X$
- $\mathit{Pos}(F)$: $X \stackrel{\min}{=} F \vee \langle \mathit{Act} \rangle X$

- $\mathit{Safe}(F)$: $X \stackrel{\max}{=} F \wedge ([\mathit{Act}]\mathit{ff} \vee \langle \mathit{Act} \rangle X)$
- $\mathit{Even}(F)$: $X \stackrel{\min}{=} F \vee (\langle \mathit{Act} \rangle \mathit{tt} \wedge [\mathit{Act}]X)$

- $F\ \mathcal{U}^w\ G$: $X \stackrel{\max}{=} G \vee (F \wedge [\mathit{Act}]X)$
- $F\ \mathcal{U}^s\ G$: $X \stackrel{\min}{=} G \vee (F \wedge \langle \mathit{Act} \rangle \mathit{tt} \wedge [\mathit{Act}]X)$

Using until we can express e.g. $\mathit{Inv}(F)$ and $\mathit{Even}(F)$:

$$\mathit{Inv}(F) \equiv F\ \mathcal{U}^w\ \mathit{ff} \qquad \mathit{Even}(F) \equiv \mathit{tt}\ \mathcal{U}^s\ F$$

Selection of Temporal Properties

- *Inv(F)*: $X \stackrel{\max}{=} F \wedge [Act]X$
- *Pos(F)*: $X \stackrel{\min}{=} F \vee \langle Act \rangle X$

- *Safe(F)*: $X \stackrel{\max}{=} F \wedge ([Act]\mathit{ff} \vee \langle Act \rangle X)$
- *Even(F)*: $X \stackrel{\min}{=} F \vee (\langle Act \rangle \mathit{tt} \wedge [Act]X)$

- $F\ \mathcal{U}^w\ G$: $X \stackrel{\max}{=} G \vee (F \wedge [Act]X)$
- $F\ \mathcal{U}^s\ G$: $X \stackrel{\min}{=} G \vee (F \wedge \langle Act \rangle \mathit{tt} \wedge [Act]X)$

Using until we can express e.g. *Inv(F)* and *Even(F)*:

$$Inv(F) \equiv F\ \mathcal{U}^w\ \mathit{ff} \qquad\qquad Even(F) \equiv \mathit{tt}\ \mathcal{U}^s\ F$$

Examples of More Advanced Recursive Formulae

Nested Definitions of Recursive Variables

$$X \stackrel{\min}{=} Y \vee \langle Act \rangle X \qquad\qquad Y \stackrel{\max}{=} \langle a \rangle tt \wedge \langle Act \rangle Y$$

Solution: compute first $[\![Y]\!]$ and then $[\![X]\!]$.

Mutually Recursive Definitions

$$X \stackrel{\max}{=} [a]Y \qquad\qquad Y \stackrel{\max}{=} \langle a \rangle X$$

Solution: consider a complete lattice $(2^{Proc} \times 2^{Proc}, \sqsubseteq)$ where $(S_1, S_2) \sqsubseteq (S_1', S_2')$ iff $S_1 \subseteq S_1'$ and $S_2 \subseteq S_2'$.

Theorem (Characteristic Property for Finite-State Processes)

Let s be a process with finitely many reachable states. There exists a property X_s s.t. for all processes t: $s \sim t$ if and only if $t \in [\![X_s]\!]$.

Examples of More Advanced Recursive Formulae

Nested Definitions of Recursive Variables

$$X \overset{\min}{=} Y \vee \langle Act \rangle X \qquad\qquad Y \overset{\max}{=} \langle a \rangle tt \wedge \langle Act \rangle Y$$

Solution: compute first $[\![Y]\!]$ and then $[\![X]\!]$.

Mutually Recursive Definitions

$$X \overset{\max}{=} [a]Y \qquad\qquad Y \overset{\max}{=} \langle a \rangle X$$

Solution: consider a complete lattice $(2^{Proc} \times 2^{Proc}, \sqsubseteq)$ where $(S_1, S_2) \sqsubseteq (S_1', S_2')$ iff $S_1 \subseteq S_1'$ and $S_2 \subseteq S_2'$.

Theorem (Characteristic Property for Finite-State Processes)

Let s be a process with finitely many reachable states. There exists a property X_s s.t. for all processes t: $s \sim t$ if and only if $t \in [\![X_s]\!]$.

Examples of More Advanced Recursive Formulae

Nested Definitions of Recursive Variables

$$X \stackrel{\min}{=} Y \vee \langle Act \rangle X \qquad\qquad Y \stackrel{\max}{=} \langle a \rangle tt \wedge \langle Act \rangle Y$$

Solution: compute first $[\![Y]\!]$ and then $[\![X]\!]$.

Mutually Recursive Definitions

$$X \stackrel{\max}{=} [a]Y \qquad\qquad Y \stackrel{\max}{=} \langle a \rangle X$$

Solution: consider a complete lattice $(2^{Proc} \times 2^{Proc}, \sqsubseteq)$ where $(S_1, S_2) \sqsubseteq (S_1', S_2')$ iff $S_1 \subseteq S_1'$ and $S_2 \subseteq S_2'$.

Theorem (Characteristic Property for Finite-State Processes)

Let s be a process with finitely many reachable states. There exists a property X_s s.t. for all processes t: $s \sim t$ if and only if $t \in [\![X_s]\!]$.

Definition of Strong Bisimulation

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

Strong Bisimulation

A binary relation $R \subseteq Proc \times Proc$ is a strong bisimulation iff whenever $(s, t) \in R$ then for each $a \in Act$:

- if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s', t') \in R$
- if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s', t') \in R$.

Two processes $p, q \in Proc$ are strongly bisimilar $(p \sim q)$ iff there exists a strong bisimulation R such that $(p, q) \in R$.

$$\sim \; = \bigcup \{R \mid R \text{ is a strong bisimulation}\}$$

Strong Bisimulation as a Greatest Fixed Point

Function $\mathcal{F} : 2^{(Proc \times Proc)} \to 2^{(Proc \times Proc)}$

Let $S \subseteq Proc \times Proc$. Then we define $\mathcal{F}(S)$ as follows:

$(s, t) \in \mathcal{F}(S)$ if and only if for each $a \in Act$:

- if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s', t') \in S$
- if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s', t') \in S$.

Observations

- $(2^{(Proc \times Proc)}, \subseteq)$ is a complete lattice and $\mathcal{F}$ is monotonic
- S is a strong bisimulation if and only if $S \subseteq \mathcal{F}(S)$

Strong Bisimilarity is the Greatest Fixed Point of $\mathcal{F}$

$$\sim = \bigcup \{ S \in 2^{(Proc \times Proc)} \mid S \subseteq \mathcal{F}(S) \}$$

Strong Bisimulation as a Greatest Fixed Point

Function $\mathcal{F} : 2^{(Proc \times Proc)} \to 2^{(Proc \times Proc)}$

Let $S \subseteq Proc \times Proc$. Then we define $\mathcal{F}(S)$ as follows:

$(s, t) \in \mathcal{F}(S)$ if and only if for each $a \in Act$:

- if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s', t') \in S$
- if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s', t') \in S$.

Observations

- $(2^{(Proc \times Proc)}, \subseteq)$ is a complete lattice and $\mathcal{F}$ is monotonic
- S is a strong bisimulation if and only if $S \subseteq \mathcal{F}(S)$

Strong Bisimilarity is the Greatest Fixed Point of $\mathcal{F}$

$$\sim = \bigcup \{ S \in 2^{(Proc \times Proc)} \mid S \subseteq \mathcal{F}(S) \}$$

Strong Bisimulation as a Greatest Fixed Point

Function $\mathcal{F} : 2^{(Proc \times Proc)} \rightarrow 2^{(Proc \times Proc)}$

Let $S \subseteq Proc \times Proc$. Then we define $\mathcal{F}(S)$ as follows:

$(s, t) \in \mathcal{F}(S)$ if and only if for each $a \in Act$:

- if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s', t') \in S$
- if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s', t') \in S$.

Observations

- $(2^{(Proc \times Proc)}, \subseteq)$ is a complete lattice and $\mathcal{F}$ is monotonic
- S is a strong bisimulation if and only if $S \subseteq \mathcal{F}(S)$

Strong Bisimilarity is the Greatest Fixed Point of $\mathcal{F}$

$$\sim = \bigcup \{S \in 2^{(Proc \times Proc)} \mid S \subseteq \mathcal{F}(S)\}$$

Example

Consider processes:

- $Q_1 = b.Q_2 + a.Q_3$
- $Q_2 = c.Q_4$
- $Q_3 = c.Q_4$
- $Q_4 = b.Q_2 + a.Q_3 + a.Q_1$

With the non recursive definition, in order to construct $\sim$ we had to consider that $Q_i \sim Q_i$, with $1 \leq i \leq 4$, then we had to check whether $Q_i \sim Q_j$, for all possible $i \neq j$, using the bisimulation game (and noticing that $Q_i \sim Q_j \iff Q_j \sim Q_i$).

For instance, to show that $Q_1 \not\sim Q_4$:

1. (Q_1, Q_4) **A:** $Q_4 \xrightarrow{a} Q_1$ **D:** $Q_1 \xrightarrow{a} Q_3$
2. (Q_3, Q_1) **A:** $Q_3 \xrightarrow{c} Q_4$ **D:** $Q_1 \not\xrightarrow{c}$

Example

Consider processes:

- $Q_1 = b.Q_2 + a.Q_3$
- $Q_2 = c.Q_4$
- $Q_3 = c.Q_4$
- $Q_4 = b.Q_2 + a.Q_3 + a.Q_1$

With the non recursive definition, in order to construct $\sim$ we had to consider that $Q_i \sim Q_i$, with $1 \leq i \leq 4$, then we had to check whether $Q_i \sim Q_j$, for all possible $i \neq j$, using the bisimulation game (and noticing that $Q_i \sim Q_j \iff Q_j \sim Q_i$).

For instance, to show that $Q_1 \not\sim Q_4$:

1. (Q_1, Q_4) **A:** $Q_4 \xrightarrow{a} Q_1$ **D:** $Q_1 \xrightarrow{a} Q_3$
2. (Q_3, Q_1) **A:** $Q_3 \xrightarrow{c} Q_4$ **D:** $Q_1 \not\xrightarrow{c}$

Example

Consider processes:

- $Q_1 = b.Q_2 + a.Q_3$
- $Q_2 = c.Q_4$
- $Q_3 = c.Q_4$
- $Q_4 = b.Q_2 + a.Q_3 + a.Q_1$

With the non recursive definition, in order to construct $\sim$ we had to consider that $Q_i \sim Q_i$, with $1 \leq i \leq 4$, then we had to check whether $Q_i \sim Q_j$, for all possible $i \neq j$, using the bisimulation game (and noticing that $Q_i \sim Q_j \iff Q_j \sim Q_i$).

For instance, to show that $Q_1 \not\sim Q_4$:

1. (Q_1, Q_4) **A:** $Q_4 \xrightarrow{a} Q_1$ **D:** $Q_1 \xrightarrow{a} Q_3$
2. (Q_3, Q_1) **A:** $Q_3 \xrightarrow{c} Q_4$ **D:** $Q_1 \not\xrightarrow{c}$

Example

Consider processes:

- $Q_1 = b.Q_2 + a.Q_3$
- $Q_2 = c.Q_4$
- $Q_3 = c.Q_4$
- $Q_4 = b.Q_2 + a.Q_3 + a.Q_1$

If we instead rely on the mathematical solution of the recursive definition then we have that:

$\mathcal{F}^0(\top) = \mathcal{F}^0(Proc \times Proc) = Proc \times Proc$

$\mathcal{F}^1(\top) = \mathcal{F}(Proc \times Proc) = \{(Q_1, Q_4), (Q_4, Q_1), (Q_2, Q_3), (Q_3, Q_2)\} \cup Id$

$\mathcal{F}^2(\top) = \{(Q_2, Q_3), (Q_3, Q_2)\} \cup Id$

$\mathcal{F}^3(\top) = \{(Q_2, Q_3), (Q_3, Q_2)\} \cup Id = \mathcal{F}^2(\top)$

where $Id = \{(Q_i, Q_i) \in Proc \times Proc \mid 1 \leq i \leq 4\}$

Example

Consider processes:

- $Q_1 = b.Q_2 + a.Q_3$
- $Q_2 = c.Q_4$
- $Q_3 = c.Q_4$
- $Q_4 = b.Q_2 + a.Q_3 + a.Q_1$

If we instead rely on the mathematical solution of the recursive definition then we have that:

$\mathcal{F}^0(\top) = \mathcal{F}^0(Proc \times Proc) = Proc \times Proc$
$\mathcal{F}^1(\top) = \mathcal{F}(Proc \times Proc) = \{(Q_1, Q_4), (Q_4, Q_1), (Q_2, Q_3), (Q_3, Q_2)\} \cup Id$
$\mathcal{F}^2(\top) = \{(Q_2, Q_3), (Q_3, Q_2)\} \cup Id$
$\mathcal{F}^3(\top) = \{(Q_2, Q_3), (Q_3, Q_2)\} \cup Id = \mathcal{F}^2(\top)$

where $Id = \{(Q_i, Q_i) \in Proc \times Proc \mid 1 \leq i \leq 4\}$