Modelling and Analysing Concurrent Systems

RIO 2023 Summer School of Informatics
Rio Cuarto, Argentina; February 13–17, 2023

Lecture 3: Logical Specifications

Thomas Noll
Software Modelling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-22-23/rio/

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Outline of Lecture 3

Verifying Correctness of Concurrent Systems

Equivalence-checking approach

$$\mathit{Impl} \equiv \mathit{Spec}$$

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

Verifying Correctness of Concurrent Systems

Equivalence-checking approach

$$\mathit{Impl} \equiv \mathit{Spec}$$

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

Model-checking approach

$$\mathit{Impl} \models \mathit{Prop}$$

- $\models$ is the satisfaction relation.
- *Prop* is a particular feature, often expressed via a logic, e.g., HML.
- *Prop* is a partial specification of the intended behaviour.

Motivation

Goal: check processes for simple properties

- action a is initially enabled
- action b is initially disabled
- a deadlock never occurs
- process always sends a reply after receiving a request

Motivation

Goal: check processes for simple properties

- action *a* is initially enabled
- action *b* is initially disabled
- a deadlock never occurs
- process always sends a reply after receiving a request

Approach:

- Formalisation in Hennessy-Milner Logic (HML)
- M. Hennessy, R. Milner: *On Observing Nondeterminism and Concurrency*, ICALP 1980, Springer LNCS 85, 299–309
- Checking by exploration of state space

Syntax of HML

Definition 3.1 (Syntax of HML) (Hennessy & Milner 1985)

The set *HMF* of Hennessy-Milner formulae over a set of actions *Act* is defined by the following syntax:

$$
\begin{array}{rll}
F ::= & \mathrm{tt} & \text{(true)} \\
\mid & \mathrm{ff} & \text{(false)} \\
\mid & F_1 \wedge F_2 & \text{(conjunction)} \\
\mid & F_1 \vee F_2 & \text{(disjunction)} \\
\mid & \langle \alpha \rangle F & \text{(diamond)} \\
\mid & [\alpha] F & \text{(box)}
\end{array}
$$

where $\alpha \in \mathit{Act}$.

Meaning of HML Constructs

- All processes satisfy tt.

Meaning of HML Constructs

- All processes satisfy tt.
- No process satisfies ff.

Meaning of HML Constructs

- All processes satisfy tt.
- No process satisfies ff.
- A process satisfies $F \wedge G$ iff it satisfies F and G.

Meaning of HML Constructs

- All processes satisfy tt.
- No process satisfies ff.
- A process satisfies $F \wedge G$ iff it satisfies F and G.
- A process satisfies $F \vee G$ iff it satisfies either F or G or both.

Meaning of HML Constructs

- All processes satisfy tt.
- No process satisfies ff.
- A process satisfies $F \wedge G$ iff it satisfies F and G.
- A process satisfies $F \vee G$ iff it satisfies either F or G or both.
- A process satisfies $\langle\alpha\rangle F$ for some $\alpha \in \mathit{Act}$ iff it affords an α-labelled transition to a state satisfying F (possibility).

Meaning of HML Constructs

- All processes satisfy tt.
- No process satisfies ff.
- A process satisfies $F \wedge G$ iff it satisfies F and G.
- A process satisfies $F \vee G$ iff it satisfies either F or G or both.
- A process satisfies $\langle\alpha\rangle F$ for some $\alpha \in \mathit{Act}$ iff it affords an α-labelled transition to a state satisfying F (possibility).
- A process satisfies $[\alpha]F$ for some $\alpha \in \mathit{Act}$ iff all its α-labelled transitions lead to a state satisfying F (necessity).

Meaning of HML Constructs

- All processes satisfy tt.
- No process satisfies ff.
- A process satisfies $F \wedge G$ iff it satisfies F and G.
- A process satisfies $F \vee G$ iff it satisfies either F or G or both.
- A process satisfies $\langle\alpha\rangle F$ for some $\alpha \in Act$ iff it affords an α-labelled transition to a state satisfying F (possibility).
- A process satisfies $[\alpha]F$ for some $\alpha \in Act$ iff all its α-labelled transitions lead to a state satisfying F (necessity).

Abbreviations for $L = \{\alpha_1, \ldots, \alpha_n\}$ ($n \in \mathbb{N}$):

- $\langle L\rangle F := \langle\alpha_1\rangle F \vee \ldots \vee \langle\alpha_n\rangle F$
- $[L]F := [\alpha_1]F \wedge \ldots \wedge [\alpha_n]F$
- In particular, a process satisfies
  - $\langle Act\rangle$tt iff it has some outgoing transition, i.e., is not deadlocked, and
  - $[Act]$ff iff it has no outgoing transition, i.e., is deadlocked

Semantics of HML I

Definition 3.2 (Semantics of HML)

Let $(S, Act, \longrightarrow)$ be an LTS and $F \in HMF$. The set of processes in S that satisfy F, $[\![F]\!] \subseteq S$, is defined by:

$$[\![\mathsf{tt}]\!] := S \qquad [\![\mathsf{ff}]\!] := \emptyset$$
$$[\![F_1 \wedge F_2]\!] := [\![F_1]\!] \cap [\![F_2]\!] \qquad [\![F_1 \vee F_2]\!] := [\![F_1]\!] \cup [\![F_2]\!]$$
$$[\![\langle\alpha\rangle F]\!] := \langle\cdot\alpha\cdot\rangle([\![F]\!]) \qquad [\![[\alpha]F]\!] := [\cdot\alpha\cdot]([\![F]\!])$$

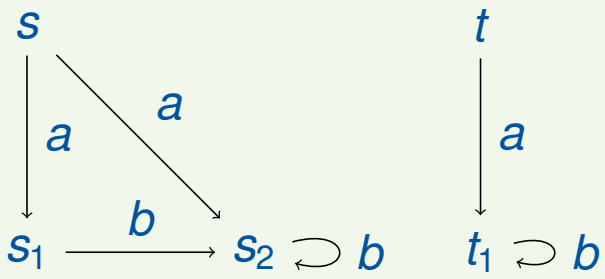
where $\langle\cdot\alpha\cdot\rangle, [\cdot\alpha\cdot] : 2^S \to 2^S$ are given by

$$\langle\cdot\alpha\cdot\rangle(T) := \{s \in S \mid \exists s' \in T : s \xrightarrow{\alpha} s'\}$$
$$[\cdot\alpha\cdot](T) := \{s \in S \mid \forall s' \in S : s \xrightarrow{\alpha} s' \Rightarrow s' \in T\}$$

We write $s \models F$ iff $s \in [\![F]\!]$. Two HML formulae are equivalent (written $F \equiv G$) iff they are satisfied by the same processes in every LTS.

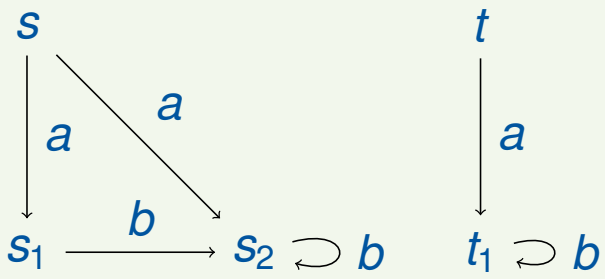
Semantics of HML II

Example 3.3 ($\langle\cdot\alpha\cdot\rangle$, $[\cdot\alpha\cdot]$ operators)



Semantics of HML II

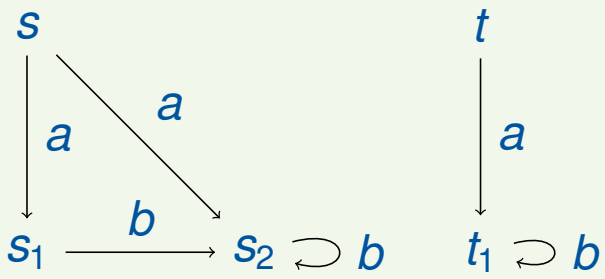
Example 3.3 ($\langle\cdot\alpha\cdot\rangle$, $[\cdot\alpha\cdot]$ operators)



(1) $\langle\cdot a\cdot\rangle(\{s_1, t_1\}) = \{s, t\}$

Semantics of HML II

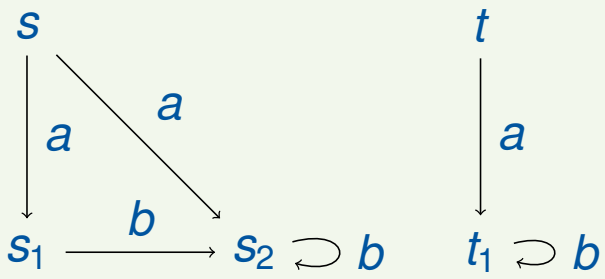
Example 3.3 ($\langle\cdot\alpha\cdot\rangle$, $[\cdot\alpha\cdot]$ operators)



(1) $\langle\cdot a\cdot\rangle(\{s_1, t_1\}) = \{s, t\}$
(2) $[\cdot a\cdot](\{s_1, t_1\}) = \{s_1, s_2, t, t_1\}$

Semantics of HML II

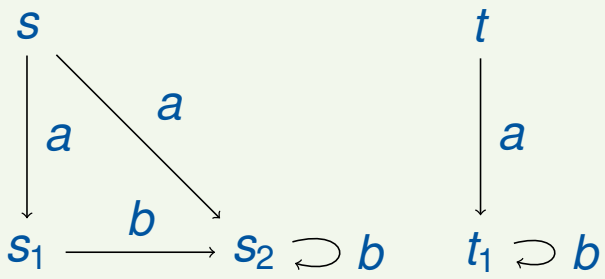
Example 3.3 ($\langle \cdot \alpha \cdot \rangle$, $[\cdot \alpha \cdot]$ operators)



(1) $\langle \cdot a \cdot \rangle(\{s_1, t_1\}) = \{s, t\}$
(2) $[\cdot a \cdot](\{s_1, t_1\}) = \{s_1, s_2, t, t_1\}$
(3) $\langle \cdot b \cdot \rangle(\{s_1, t_1\}) = \{t_1\}$

Semantics of HML II

Example 3.3 ($\langle \cdot \alpha \cdot \rangle$, $[\cdot \alpha \cdot]$ operators)



(1) $\langle \cdot a \cdot \rangle(\{s_1, t_1\}) = \{s, t\}$
(2) $[\cdot a \cdot](\{s_1, t_1\}) = \{s_1, s_2, t, t_1\}$
(3) $\langle \cdot b \cdot \rangle(\{s_1, t_1\}) = \{t_1\}$
(4) $[\cdot b \cdot](\{s_1, t_1\}) = \{s, t, t_1\}$

Simple Properties Revisited

Example 3.4

(1) Action a is initially enabled: $\langle a \rangle \mathrm{tt}$

$$\begin{aligned}[\![\langle a \rangle \mathrm{tt}]\!] &= \langle \cdot a \cdot \rangle [\![\mathrm{tt}]\!] \\ &= \langle \cdot a \cdot \rangle (S) \\ &= \{ s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \} =: \{ s \in S \mid s \xrightarrow{a} \}\end{aligned}$$

Simple Properties Revisited

Example 3.4

(1) Action a is initially enabled: $\langle a \rangle \mathrm{tt}$

$$\begin{aligned}
[\![\langle a \rangle \mathrm{tt}]\!] &= \langle \cdot a \cdot \rangle [\![\mathrm{tt}]\!] \\
&= \langle \cdot a \cdot \rangle (S) \\
&= \{ s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \} =: \{ s \in S \mid s \xrightarrow{a} \}
\end{aligned}$$

(2) Action b is initially disabled: $[b]\mathrm{ff}$

$$\begin{aligned}
[\![[b]\mathrm{ff}]\!] &= [\cdot b \cdot] [\![\mathrm{ff}]\!] \\
&= [\cdot b \cdot] (\emptyset) \\
&= \{ s \in S \mid \forall s' \in S : s \xrightarrow{b} s' \Rightarrow s' \in \emptyset \} \\
&= \{ s \in S \mid \nexists s' \in S : s \xrightarrow{b} s' \} =: \{ s \in S \mid s \not\xrightarrow{b} \}
\end{aligned}$$

Simple Properties Revisited

Example 3.4

(1) Action a is initially enabled: $\langle a \rangle \mathrm{tt}$

$$\begin{aligned}[\![\langle a \rangle \mathrm{tt}]\!] &= \langle \cdot a \cdot \rangle [\![\mathrm{tt}]\!] \\ &= \langle \cdot a \cdot \rangle (S) \\ &= \{ s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \} =: \{ s \in S \mid s \xrightarrow{a} \}\end{aligned}$$

(2) Action b is initially disabled: $[b]\mathrm{ff}$

$$\begin{aligned}[\![[b]\mathrm{ff}]\!] &= [\cdot b \cdot] [\![\mathrm{ff}]\!] \\ &= [\cdot b \cdot] (\emptyset) \\ &= \{ s \in S \mid \forall s' \in S : s \xrightarrow{b} s' \Rightarrow s' \in \emptyset \} \\ &= \{ s \in S \mid \nexists s' \in S : s \xrightarrow{b} s' \} =: \{ s \in S \mid s \not\xrightarrow{b} \}\end{aligned}$$

(3) Absence of deadlock:

- initially: $\langle Act \rangle \mathrm{tt}$
- always (i.e., no reachable state deadlocks): later (requires recursion)

Simple Properties Revisited

Example 3.4

(1) Action a is initially enabled: $\langle a \rangle \mathrm{tt}$

$$\begin{aligned}[\![\langle a \rangle \mathrm{tt}]\!] &= \langle \cdot a \cdot \rangle [\![\mathrm{tt}]\!] \\ &= \langle \cdot a \cdot \rangle (S) \\ &= \{ s \in S \mid \exists s' \in S : s \xrightarrow{a} s' \} =: \{ s \in S \mid s \xrightarrow{a} \}\end{aligned}$$

(2) Action b is initially disabled: $[b]\mathrm{ff}$

$$\begin{aligned}[\![[b]\mathrm{ff}]\!] &= [\cdot b \cdot] [\![\mathrm{ff}]\!] \\ &= [\cdot b \cdot] (\emptyset) \\ &= \{ s \in S \mid \forall s' \in S : s \xrightarrow{b} s' \Rightarrow s' \in \emptyset \} \\ &= \{ s \in S \mid \nexists s' \in S : s \xrightarrow{b} s' \} =: \{ s \in S \mid s \not\xrightarrow{b} \}\end{aligned}$$

(3) Absence of deadlock:
- initially: $\langle Act \rangle \mathrm{tt}$
- always (i.e., no reachable state deadlocks): later (requires recursion)

(4) Responsiveness:
- initially: $[request] \langle \overline{reply} \rangle \mathrm{tt}$
- always: later (requires recursion)

Outline of Lecture 3

HML and Process Traces I

Lemma 3.5 (HML and process traces)

Let $(S, Act, \longrightarrow)$ *be an LTS, and let* $P, Q \in S$ *satisfy the same HMF (i.e., for all* $F \in HMF$*:* $P \models F \iff Q \models F$*). Then* $Tr(P) = Tr(Q)$*.*

HML and Process Traces I

Lemma 3.5 (HML and process traces)

Let $(S, Act, \longrightarrow)$ be an LTS, and let $P, Q \in S$ satisfy the same HMF (i.e., for all $F \in HMF$: $P \models F \iff Q \models F$). Then $Tr(P) = Tr(Q)$.

Proof.

Let $P, Q \in S$ such that for all $F \in HMF$: $P \models F \iff Q \models F$.

HML and Process Traces I

Lemma 3.5 (HML and process traces)

Let $(S, Act, \longrightarrow)$ be an LTS, and let $P, Q \in S$ satisfy the same HMF (i.e., for all $F \in HMF$: $P \models F \iff Q \models F$). Then $Tr(P) = Tr(Q)$.

Proof.

Let $P, Q \in S$ such that for all $F \in HMF$: $P \models F \iff Q \models F$.

Assumption: $Tr(P) \neq Tr(Q)$.

HML and Process Traces I

Lemma 3.5 (HML and process traces)

Let $(S, Act, \longrightarrow)$ be an LTS, and let $P, Q \in S$ satisfy the same HMF (i.e., for all $F \in HMF$: $P \models F \iff Q \models F$). Then $Tr(P) = Tr(Q)$.

Proof.

Let $P, Q \in S$ such that for all $F \in HMF$: $P \models F \iff Q \models F$.

Assumption: $Tr(P) \neq Tr(Q)$.

Then there ex. $n \geq 1$ and $w = \alpha_1 \dots \alpha_n \in Act^+$ (note that $\varepsilon \in Tr(P)$ for all $R \in S$) such that $w \in Tr(P) \setminus Tr(Q)$ (or vice versa).
Hence, for $F := \langle \alpha_1 \rangle \dots \langle \alpha_n \rangle \mathrm{tt} \in HMF$: $P \models F$ but $Q \not\models F$. ↯ □

HML and Process Traces II

Remark: the converse does *not* hold.

Example 3.6

- Let
  - $P := a.(b.\text{nil} + c.\text{nil}) \in Prc$ and
  - $Q := a.b.\text{nil} + a.c.\text{nil} \in Prc$
- Then $Tr(P) = Tr(Q) = \{\varepsilon, a, ab, ac\}$

Remark: the converse does *not* hold.

Example 3.6

- Let
  - $P := a.(b.\text{nil} + c.\text{nil}) \in Prc$ and
  - $Q := a.b.\text{nil} + a.c.\text{nil} \in Prc$
- Then $Tr(P) = Tr(Q) = \{\varepsilon, a, ab, ac\}$
- Let $F := [a](\langle b\rangle \text{tt} \wedge \langle c\rangle \text{tt}) \in HMF$
- Then $P \models F$ but $Q \not\models F$

HML and Process Traces II

Remark: the converse does *not* hold.

Example 3.6

- Let
  - $P := a.(b.\text{nil} + c.\text{nil}) \in Prc$ and
  - $Q := a.b.\text{nil} + a.c.\text{nil} \in Prc$
- Then $Tr(P) = Tr(Q) = \{\varepsilon, a, ab, ac\}$
- Let $F := [a](\langle b\rangle \text{tt} \wedge \langle c\rangle \text{tt}) \in HMF$
- Then $P \models F$ but $Q \not\models F$
- Thus: HML can distinguish branching behaviour of processes.

Outline of Lecture 3

Bisimilarity and HML

- Strong bisimilarity (and observation congruence) are based on mutually mimicking of processes.
- They possess the required properties of behavioural equivalences.
- In particular, $\sim$ and $\approx^c$ are deadlock-sensitive CCS congruences.
- Hennessy-Milner Logic (HML) is a logic for expressing properties of processes.

Bisimilarity and HML

- Strong bisimilarity (and observation congruence) are based on mutually mimicking of processes.
- They possess the required properties of behavioural equivalences.
- In particular, $\sim$ and $\approx^c$ are deadlock-sensitive CCS congruences.
- Hennessy-Milner Logic (HML) is a logic for expressing properties of processes.

Aim

Study the connection between strong bisimilarity and satisfaction of HML formulae.

Relationship Between HML and Strong Bisimilarity

(ignoring a technical side condition called “finitely branching”)

Theorem 3.7 (Hennessy-Milner Theorem)

Let $(S, Act, \{\xrightarrow{a} \mid a \in Act\})$ *be an LTS and* $P, Q \in S$*. Then:*

$$P \sim Q \quad \textit{iff} \quad \textit{for every } F \in HMF : (P \models F \iff Q \models F).$$

Proof.

omitted □

Proving Non-Bisimilarity

Proving non-bisimilarity

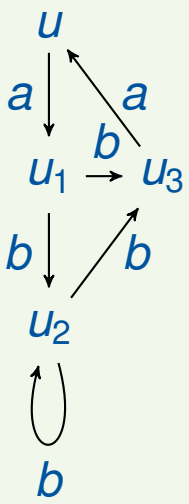
Showing $P \not\sim Q$ thus amounts to finding a single HML-formula F with $P \models F$ and $Q \not\models F$.

Proving Non-Bisimilarity

Proving non-bisimilarity

Showing $P \not\sim Q$ thus amounts to finding a single HML-formula F with $P \models F$ and $Q \not\models F$.

Example 3.8

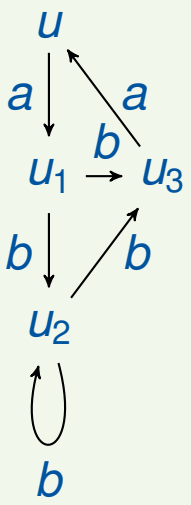


Proving Non-Bisimilarity

Proving non-bisimilarity

Showing $P \not\sim Q$ thus amounts to finding a single HML-formula F with $P \models F$ and $Q \not\models F$.

Example 3.8



Distinguishing formulae

(satisfied by row state / violated by column state):

	u	u_1	u_2	u_3
u	-	$\langle a \rangle \mathrm{tt}$	$\langle a \rangle \mathrm{tt}$	$\langle a \rangle \langle b \rangle \mathrm{tt}$
u_1	$[a]\mathrm{ff}$	-	-	$[a]\mathrm{ff}$
u_2	$[a]\mathrm{ff}$	-	-	$[a]\mathrm{ff}$
u_3	$\langle a \rangle \langle a \rangle \mathrm{tt}$	$\langle a \rangle \mathrm{tt}$	$\langle a \rangle \mathrm{tt}$	-

(check using CAAL)

Outline of Lecture 3

Finiteness of HML

Observation: HML formulae only describe finite part of process behaviour

- each modal operator ($[.]$, $\langle . \rangle$) talks about one step
- only finite nesting of operators ("modal depth")

Finiteness of HML

Observation: HML formulae only describe finite part of process behaviour

- each modal operator ($[.]$, $\langle . \rangle$) talks about one step
- only finite nesting of operators ("modal depth")

Example 3.9

- $F := (\langle a \rangle [a]\mathrm{ff}) \vee \langle b \rangle \mathrm{tt} \in \mathit{HMF}$ has modal depth 2
- Checking F involves analysis of all behaviours of length ≤ 2

Finiteness of HML

Observation: HML formulae only describe finite part of process behaviour

- each modal operator ($[.]$, $\langle . \rangle$) talks about one step
- only finite nesting of operators ("modal depth")

Example 3.9

- $F := (\langle a \rangle [a]\mathrm{ff}) \vee \langle b \rangle \mathrm{tt} \in \mathit{HMF}$ has modal depth 2
- Checking F involves analysis of all behaviours of length ≤ 2

But: sometimes necessary to refer to arbitrarily long computations
(e.g., "no deadlock state reachable")

- possible solution: support infinite conjunctions and disjunctions

Infinite Conjunctions and Disjunctions

Example 3.10

(1) Invariant properties

- e.g., “action a is enabled in every reachable state”
- that is, every reachable state satisfies $\langle a \rangle \mathrm{tt}$
- requires infinite conjunction:

$$Inv(\langle a \rangle \mathrm{tt}) = \langle a \rangle \mathrm{tt} \wedge [Act]\langle a \rangle \mathrm{tt} \wedge [Act][Act]\langle a \rangle \mathrm{tt} \wedge \ldots = \bigwedge_{k \in \mathbb{N}} [Act]^k \langle a \rangle \mathrm{tt}$$

Infinite Conjunctions and Disjunctions

Example 3.10

(1) Invariant properties

- e.g., “action a is enabled in every reachable state”
- that is, every reachable state satisfies $\langle a \rangle \mathrm{tt}$
- requires infinite conjunction:

$$Inv(\langle a \rangle \mathrm{tt}) = \langle a \rangle \mathrm{tt} \wedge [Act]\langle a \rangle \mathrm{tt} \wedge [Act][Act]\langle a \rangle \mathrm{tt} \wedge \ldots = \bigwedge_{k \in \mathbb{N}} [Act]^k \langle a \rangle \mathrm{tt}$$

(2) Possibility properties

- e.g., “the process has the possibility to terminate”
- that is, it has the option to eventually satisfy $[Act]\mathrm{ff}$
- Representable by infinite disjunction:

$$Pos([Act]\mathrm{ff}) = [Act]\mathrm{ff} \vee \langle Act \rangle [Act]\mathrm{ff} \vee \langle Act \rangle \langle Act \rangle [Act]\mathrm{ff} \vee \ldots = \bigvee_{k \in \mathbb{N}} \langle Act \rangle^k [Act]\mathrm{ff}$$

Infinite Conjunctions and Disjunctions

Example 3.10

(1) Invariant properties

- e.g., “action a is enabled in every reachable state”
- that is, every reachable state satisfies $\langle a \rangle \mathrm{tt}$
- requires infinite conjunction:

$$Inv(\langle a \rangle \mathrm{tt}) = \langle a \rangle \mathrm{tt} \wedge [Act]\langle a \rangle \mathrm{tt} \wedge [Act][Act]\langle a \rangle \mathrm{tt} \wedge \ldots = \bigwedge_{k \in \mathbb{N}} [Act]^k \langle a \rangle \mathrm{tt}$$

(2) Possibility properties

- e.g., “the process has the possibility to terminate”
- that is, it has the option to eventually satisfy $[Act]\mathrm{ff}$
- Representable by infinite disjunction:

$$Pos([Act]\mathrm{ff}) = [Act]\mathrm{ff} \vee \langle Act \rangle [Act]\mathrm{ff} \vee \langle Act \rangle \langle Act \rangle [Act]\mathrm{ff} \vee \ldots = \bigvee_{k \in \mathbb{N}} \langle Act \rangle^k [Act]\mathrm{ff}$$

Problem: infinite formulae not easy to handle!

Introducing Recursion

Solution: employ recursion!

- $\mathit{Inv}(\langle a \rangle \mathrm{tt}) = \langle a \rangle \mathrm{tt} \wedge [\mathit{Act}]\, \mathit{Inv}(\langle a \rangle \mathrm{tt})$
- $\mathit{Pos}([\mathit{Act}]\mathrm{ff}) = [\mathit{Act}]\mathrm{ff} \vee \langle \mathit{Act} \rangle\, \mathit{Pos}([\mathit{Act}]\mathrm{ff})$

Introducing Recursion

Solution: employ recursion!

- $\mathit{Inv}(\langle a \rangle \mathrm{tt}) = \langle a \rangle \mathrm{tt} \wedge [\mathit{Act}]\, \mathit{Inv}(\langle a \rangle \mathrm{tt})$
- $\mathit{Pos}([\mathit{Act}]\mathrm{ff}) = [\mathit{Act}]\mathrm{ff} \vee \langle \mathit{Act} \rangle\, \mathit{Pos}([\mathit{Act}]\mathrm{ff})$

Interpretation: the sets of states $X, Y \subseteq S$ satisfying the respective formula should solve the corresponding semantic equations, i.e.,

- $X = \langle \cdot a \cdot \rangle(S) \cap [\cdot \mathit{Act} \cdot](X)$
- $Y = [\cdot \mathit{Act} \cdot](\emptyset) \cup \langle \cdot \mathit{Act} \cdot \rangle(Y)$

Introducing Recursion

Solution: employ recursion!

- $\mathit{Inv}(\langle a \rangle \mathrm{tt}) = \langle a \rangle \mathrm{tt} \wedge [\mathit{Act}]\, \mathit{Inv}(\langle a \rangle \mathrm{tt})$
- $\mathit{Pos}([\mathit{Act}]\mathrm{ff}) = [\mathit{Act}]\mathrm{ff} \vee \langle \mathit{Act} \rangle\, \mathit{Pos}([\mathit{Act}]\mathrm{ff})$

Interpretation: the sets of states $X, Y \subseteq S$ satisfying the respective formula should solve the corresponding semantic equations, i.e.,

- $X = \langle \cdot a \cdot \rangle(S) \cap [\cdot \mathit{Act} \cdot](X)$
- $Y = [\cdot \mathit{Act} \cdot](\emptyset) \cup \langle \cdot \mathit{Act} \cdot \rangle(Y)$

Open questions

- Do such recursive equations (always) have solutions?
- If so, are these unique?
- How can we decide whether a process satisfies a recursive formula (“model checking”)?

Introducing Recursion

Solution: employ recursion!

- $\mathit{Inv}(\langle a \rangle \mathrm{tt}) = \langle a \rangle \mathrm{tt} \wedge [\mathit{Act}]\, \mathit{Inv}(\langle a \rangle \mathrm{tt})$
- $\mathit{Pos}([\mathit{Act}]\mathrm{ff}) = [\mathit{Act}]\mathrm{ff} \vee \langle \mathit{Act} \rangle\, \mathit{Pos}([\mathit{Act}]\mathrm{ff})$

Interpretation: the sets of states $X, Y \subseteq S$ satisfying the respective formula should solve the corresponding semantic equations, i.e.,

- $X = \langle \cdot a \cdot \rangle(S) \cap [\cdot \mathit{Act} \cdot](X)$
- $Y = [\cdot \mathit{Act} \cdot](\emptyset) \cup \langle \cdot \mathit{Act} \cdot \rangle(Y)$

Open questions

- Do such recursive equations (always) have solutions?
  Yes, they do.
- If so, are these unique?
  Not necessarily.
- How can we decide whether a process satisfies a recursive formula (“model checking”)?
  Employ fixed-point iteration.

Uniqueness of Solutions

Uniqueness of solutions

- Use greatest solutions for properties that hold unless the process has a finite computation that disproves it.
- Use least solutions for properties that hold if the process has a finite computation that proves it.

Uniqueness of Solutions

Uniqueness of solutions

- Use greatest solutions for properties that hold unless the process has a finite computation that disproves it.
- Use least solutions for properties that hold if the process has a finite computation that proves it.

Example 3.11

Let $(S, Act, \longrightarrow)$ be an LTS, $s \in S$, and $F \in HMF$.

- Invariant: $Inv(F) \equiv X$ for $X \overset{max}{=} F \wedge [Act]X$
  - $s \models Inv(F)$ if all states reachable from s satisfy F
- Possibility: $Pos(F) \equiv Y$ for $Y \overset{min}{=} F \vee \langle Act \rangle Y$
  - $s \models Pos(F)$ if a state satisfying F is reachable from s

Outline of Lecture 3

Fixed-Point Iteration

- **Approach:** iteratively compute solution of equation (system)
  - Arnd's course: value iteration for linear equation systems
  - numerical analysis: Newton's method for computing roots of real-valued functions
  - ...

Fixed-Point Iteration

- **Approach:** iteratively compute solution of equation (system)
  - Arnd's course: value iteration for linear equation systems
  - numerical analysis: Newton's method for computing roots of real-valued functions
  - ...
- **"Classical" issues:**
  - convergence
  - continuous domains approximated by floating-point numbers (rounding errors)
  - ...

Fixed-Point Iteration

- **Approach:** iteratively compute solution of equation (system)
  - Arnd's course: value iteration for linear equation systems
  - numerical analysis: Newton's method for computing roots of real-valued functions
  - ...
- **"Classical" issues:**
  - convergence
  - continuous domains approximated by floating-point numbers (rounding errors)
  - ...
- **Here:** much nicer setting!
  - finite (and thus discrete) domain (subsets of states)
  - monotonic behaviour of iteration functions (w.r.t. subset inclusion)
  - thus termination guaranteed

Fixed-Point Iteration

- **Approach:** iteratively compute solution of equation (system)
  - Arnd's course: value iteration for linear equation systems
  - numerical analysis: Newton's method for computing roots of real-valued functions
  - ...
- **"Classical" issues:**
  - convergence
  - continuous domains approximated by floating-point numbers (rounding errors)
  - ...
- **Here:** much nicer setting!
  - finite (and thus discrete) domain (subsets of states)
  - monotonic behaviour of iteration functions (w.r.t. subset inclusion)
  - thus termination guaranteed
- **Two variants:**

Greatest solution:

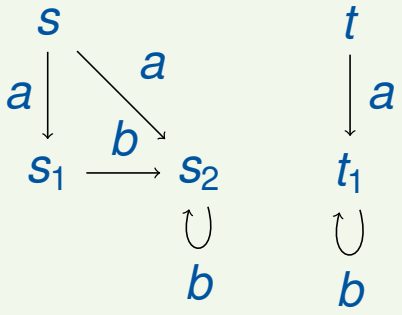
- start with greatest element (= S)
- iteration yields decreasing chain
- stable result = greatest fixed point

Least solution:

- start with least element (= $\emptyset$)
- iteration yields increasing chain
- stable result = least fixed point

A Greatest Fixed Point

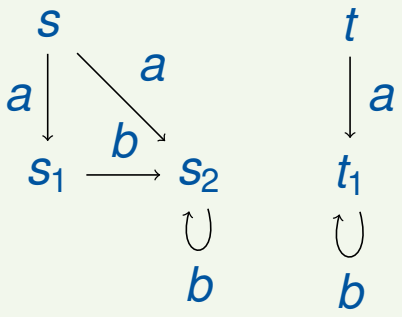
Example 3.12



Let $S := \{s, s_1, s_2, t, t_1\}$.

A Greatest Fixed Point

Example 3.12



Let $S := \{s, s_1, s_2, t, t_1\}$.

Solution of

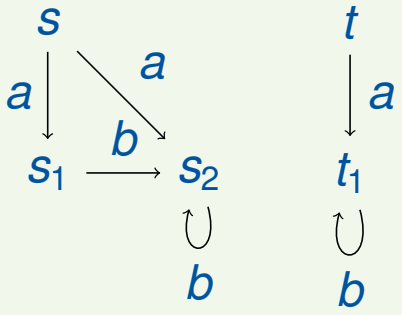
$$X \overset{max}{=} \langle b \rangle \mathrm{tt} \wedge [b]X$$

(invariant: “all b^*-successors have a b-successor”) equals the greatest fixed point of

$$f : 2^S \to 2^S : T \mapsto \langle \cdot b \cdot \rangle(S) \cap [\cdot b \cdot](T)$$

A Greatest Fixed Point

Example 3.12



Let $S := \{s, s_1, s_2, t, t_1\}$.

Solution of

$$X \overset{max}{=} \langle b \rangle \mathrm{tt} \wedge [b]X$$

(invariant: "all b^*-successors have a b-successor") equals the greatest fixed point of

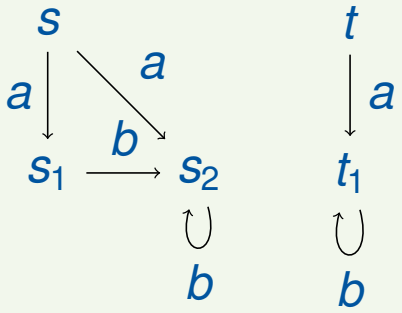
$$f : 2^S \to 2^S : T \mapsto \langle \cdot b \cdot \rangle(S) \cap [\cdot b \cdot](T)$$

Fixed-point iteration starting from greatest element (S):

$$\begin{aligned} f(S) &= \langle \cdot b \cdot \rangle(S) \cap [\cdot b \cdot](S) \\ &= \{s_1, s_2, t_1\} \cap S \\ &= \{s_1, s_2, t_1\} \end{aligned}$$

A Greatest Fixed Point

Example 3.12



Let $S := \{s, s_1, s_2, t, t_1\}$.

Solution of

$$X \overset{max}{=} \langle b \rangle \mathrm{tt} \wedge [b]X$$

(invariant: "all b^*-successors have a b-successor") equals the greatest fixed point of

$$f : 2^S \to 2^S : T \mapsto \langle \cdot b \cdot \rangle(S) \cap [\cdot b \cdot](T)$$

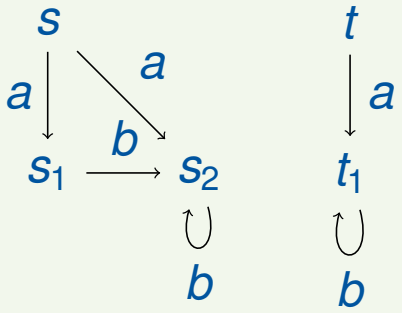
Fixed-point iteration starting from greatest element (S):

$$\begin{aligned} f(S) &= \langle \cdot b \cdot \rangle(S) \cap [\cdot b \cdot](S) \\ &= \{s_1, s_2, t_1\} \cap S \\ &= \{s_1, s_2, t_1\} \end{aligned}$$

$$\begin{aligned} f^2(S) &= \langle \cdot b \cdot \rangle(S) \cap [\cdot b \cdot](\{s_1, s_2, t_1\}) \\ &= \{s_1, s_2, t_1\} \cap \{s, s_1, s_2, t, t_1\} \\ &= \{s_1, s_2, t_1\} \\ &= f(S) \end{aligned}$$

A Greatest Fixed Point

Example 3.12



Let $S := \{s, s_1, s_2, t, t_1\}$.

Solution of

$$X \overset{max}{=} \langle b \rangle \mathrm{tt} \wedge [b]X$$

(invariant: "all b^*-successors have a b-successor") equals the greatest fixed point of

$$f : 2^S \to 2^S : T \mapsto \langle \cdot b \cdot \rangle(S) \cap [\cdot b \cdot](T)$$

Fixed-point iteration starting from greatest element (S):

$$\begin{aligned} f(S) &= \langle \cdot b \cdot \rangle(S) \cap [\cdot b \cdot](S) \\ &= \{s_1, s_2, t_1\} \cap S \\ &= \{s_1, s_2, t_1\} \end{aligned}$$

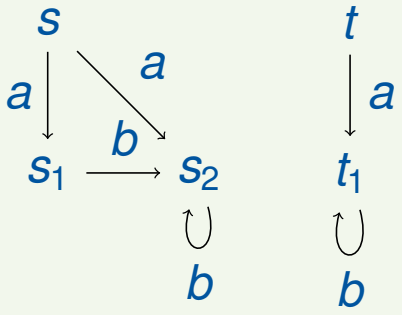
$$\begin{aligned} f^2(S) &= \langle \cdot b \cdot \rangle(S) \cap [\cdot b \cdot](\{s_1, s_2, t_1\}) \\ &= \{s_1, s_2, t_1\} \cap \{s, s_1, s_2, t, t_1\} \\ &= \{s_1, s_2, t_1\} \\ &= f(S) \end{aligned}$$

$$\Rightarrow \text{greatest solution} = \{s_1, s_2, t_1\}$$

(verify using CAAL)

A Least Fixed Point

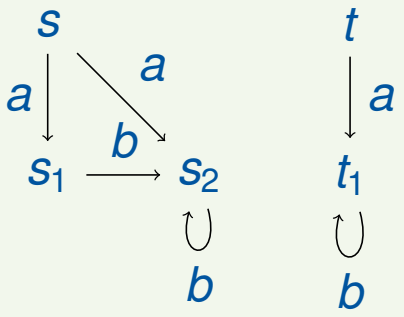
Example 3.13



Let $S := \{s, s_1, s_2, t, t_1\}$.

A Least Fixed Point

Example 3.13



Let $S := \{s, s_1, s_2, t, t_1\}$.

Solution of

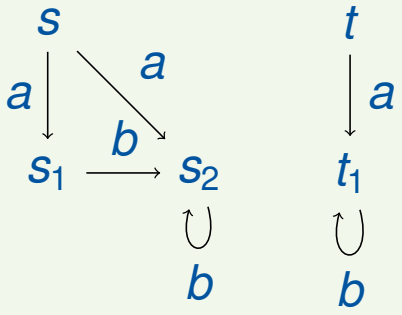
$$Y \overset{min}{=} \langle b \rangle \mathrm{tt} \vee \langle \{a, b\} \rangle Y$$

(possibility: “a b-transition is reachable”) equals the least fixed point of

$$g : 2^S \to 2^S : T \mapsto \langle \cdot b \cdot \rangle(S) \cup \langle \cdot \{a, b\} \cdot \rangle(T)$$

A Least Fixed Point

Example 3.13



Let $S := \{s, s_1, s_2, t, t_1\}$.

Solution of

$$Y \overset{min}{=} \langle b \rangle \mathrm{tt} \vee \langle \{a, b\} \rangle Y$$

(possibility: “a b-transition is reachable”) equals the least fixed point of

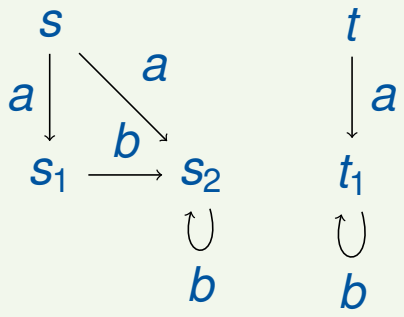
$$g : 2^S \to 2^S : T \mapsto \langle \cdot b \cdot \rangle(S) \cup \langle \cdot \{a, b\} \cdot \rangle(T)$$

Fixed-point iteration starting from least element ($\emptyset$):

$$\begin{aligned} g(\emptyset) &= \langle \cdot b \cdot \rangle(S) \cup \langle \cdot \{a, b\} \cdot \rangle(\emptyset) \\ &= \{s_1, s_2, t_1\} \cup \emptyset \\ &= \{s_1, s_2, t_1\} \end{aligned}$$

A Least Fixed Point

Example 3.13



Let $S := \{s, s_1, s_2, t, t_1\}$.

Solution of

$$Y \overset{min}{=} \langle b \rangle \mathrm{tt} \vee \langle \{a, b\} \rangle Y$$

(possibility: “a b-transition is reachable”) equals the least fixed point of

$$g : 2^S \to 2^S : T \mapsto \langle \cdot b \cdot \rangle(S) \cup \langle \cdot \{a, b\} \cdot \rangle(T)$$

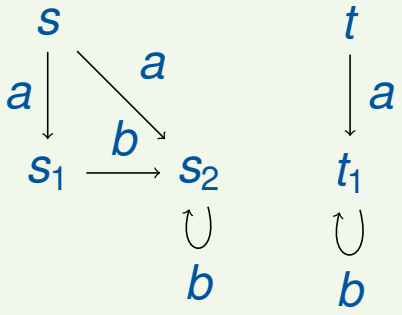
Fixed-point iteration starting from least element ($\emptyset$):

$$\begin{aligned} g(\emptyset) &= \langle \cdot b \cdot \rangle(S) \cup \langle \cdot \{a, b\} \cdot \rangle(\emptyset) \\ &= \{s_1, s_2, t_1\} \cup \emptyset \\ &= \{s_1, s_2, t_1\} \end{aligned}$$

$$\begin{aligned} g^2(\emptyset) &= \langle \cdot b \cdot \rangle(S) \cup \langle \cdot \{a, b\} \cdot \rangle(\{s_1, s_2, t_1\}) \\ &= \{s_1, s_2, t_1\} \cup \{s, s_1, s_2, t, t_1\} \\ &= \{s, s_1, s_2, t, t_1\} \\ &= S \end{aligned}$$

A Least Fixed Point

Example 3.13



Let $S := \{s, s_1, s_2, t, t_1\}$.

Solution of

$$Y \overset{min}{=} \langle b \rangle \mathrm{tt} \vee \langle \{a, b\} \rangle Y$$

(possibility: “a b-transition is reachable”) equals the least fixed point of

$$g : 2^S \to 2^S : T \mapsto \langle \cdot b \cdot \rangle(S) \cup \langle \cdot \{a, b\} \cdot \rangle(T)$$

Fixed-point iteration starting from least element ($\emptyset$):

$$\begin{aligned} g(\emptyset) &= \langle \cdot b \cdot \rangle(S) \cup \langle \cdot \{a, b\} \cdot \rangle(\emptyset) \\ &= \{s_1, s_2, t_1\} \cup \emptyset \\ &= \{s_1, s_2, t_1\} \end{aligned}$$

$$\begin{aligned} g^2(\emptyset) &= \langle \cdot b \cdot \rangle(S) \cup \langle \cdot \{a, b\} \cdot \rangle(\{s_1, s_2, t_1\}) \\ &= \{s_1, s_2, t_1\} \cup \{s, s_1, s_2, t, t_1\} \\ &= \{s, s_1, s_2, t, t_1\} \\ &= S \end{aligned}$$

$\Rightarrow$ least solution $= S$

(verify using CAAL)