



Concurrency Theory

Winter Semester 2017/18

Lecture 4: Hennessy-Milner Logic with Recursion

Joost-Pieter Katoen and Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<http://moves.rwth-aachen.de/teaching/ws-1718/ct/>

Recap: Hennessy-Milner Logic

Outline of Lecture 4

Recap: Hennessy-Milner Logic

Adding Recursion to HML

HML with One Recursive Variable

Complete Lattices

Recap: Hennessy-Milner Logic

Syntax of HML

Definition (Syntax of HML)

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

$F ::= tt$	(true)
ff	(false)
$F_1 \wedge F_2$	(conjunction)
$F_1 \vee F_2$	(disjunction)
$\langle \alpha \rangle F$	(diamond)
$[\alpha] F$	(box)

where $\alpha \in Act$.

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

- $\langle L \rangle F := \langle \alpha_1 \rangle F \vee \dots \vee \langle \alpha_n \rangle F$
- $[L] F := [\alpha_1] F \wedge \dots \wedge [\alpha_n] F$
- In particular, $\langle \emptyset \rangle F := ff$ and $[\emptyset] F := tt$

Recap: Hennessy-Milner Logic

Semantics of HML

Definition (Semantics of HML)

Let $(S, Act, \longrightarrow)$ be an LTS and $F \in HMF$. The set of processes in S that **satisfy** F ,

$\llbracket F \rrbracket \subseteq S$, is defined by:

$$\begin{aligned} \llbracket tt \rrbracket &:= S & \llbracket ff \rrbracket &:= \emptyset \\ \llbracket F_1 \wedge F_2 \rrbracket &:= \llbracket F_1 \rrbracket \cap \llbracket F_2 \rrbracket & \llbracket F_1 \vee F_2 \rrbracket &:= \llbracket F_1 \rrbracket \cup \llbracket F_2 \rrbracket \\ \llbracket \langle \alpha \rangle F \rrbracket &:= \langle \cdot \alpha \cdot \rangle (\llbracket F \rrbracket) & \llbracket [\alpha] F \rrbracket &:= [\cdot \alpha \cdot] (\llbracket F \rrbracket) \end{aligned}$$

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

$$\begin{aligned} \langle \cdot \alpha \cdot \rangle (T) &:= \{s \in S \mid \exists s' \in T : s \xrightarrow{\alpha} s'\} \\ [\cdot \alpha \cdot] (T) &:= \{s \in S \mid \forall s' \in S : s \xrightarrow{\alpha} s' \implies s' \in T\} \end{aligned}$$

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

Recap: Hennessy-Milner Logic

Process Traces

Goal: reduce processes to the action sequences they can perform

Definition (Trace language)

For every $P \in Prc$, let

$$Tr(P) := \{w \in Act^* \mid \text{ex. } P' \in Prc \text{ such that } P \xrightarrow{w} P'\}$$

be the **trace language** of P (where $\xrightarrow{w} := \xrightarrow{a_1} \circ \dots \circ \xrightarrow{a_n}$ for $w = a_1 \dots a_n$).

$P, Q \in Prc$ are called **trace equivalent** if $Tr(P) = Tr(Q)$.

Example (One-place buffer)

$$B = in.\overline{out}.B$$

$$\implies Tr(B) = (in \cdot \overline{out})^* \cdot (in + \varepsilon)$$

Recap: Hennessy-Milner Logic

HML and Process Traces

Lemma

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

Proof.

on the board □

Remark: the converse does *not* hold.

Example

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

Adding Recursion to HML

Outline of Lecture 4

Recap: Hennessy-Milner Logic

Adding Recursion to HML

HML with One Recursive Variable

Complete Lattices

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**)

Adding Recursion to HML

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 4.1

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

Adding Recursion to HML

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 4.1

- $F := (\langle a \rangle [a]ff) \vee \langle b \rangle tt \in HMF$ 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

Example 4.2

- Let $C = a.C$, $D = a.D + a.nil$
- Then $C \models [a]\langle a \rangle tt$ but $D \not\models [a]\langle a \rangle tt$

Infinite Conjunctions

Example 4.2

- Let $C = a.C$, $D = a.D + a.nil$
- Then $C \models [a]\langle a \rangle tt$ but $D \not\models [a]\langle a \rangle tt$
- Now redefine D as $D_n = a.D_n + a.E_n$ where $n \in \mathbb{N}$, $E_k = a.E_{k-1}$ ($1 \leq k \leq n$), $E_0 = nil$
- Then (for $[a]^k F := \underbrace{[a] \dots [a]}_{k \text{ times}} F$ where $F \in HMF$):
 - $C \models [a]^k \langle a \rangle tt$ for all $k \in \mathbb{N}$
 - $D_n \models [a]^k \langle a \rangle tt$ for all $0 \leq k \leq n$
 - $D_n \not\models [a]^k \langle a \rangle tt$ for all $k > n$

Adding Recursion to HML

Infinite Conjunctions

Example 4.2

- Let $C = a.C$, $D = a.D + a.nil$
- Then $C \models [a]\langle a \rangle tt$ but $D \not\models [a]\langle a \rangle tt$
- Now redefine D as $D_n = a.D_n + a.E_n$ where $n \in \mathbb{N}$, $E_k = a.E_{k-1}$ ($1 \leq k \leq n$), $E_0 = nil$
- Then (for $[a]^k F := \underbrace{[a] \dots [a]}_{k \text{ times}} F$ where $F \in HMF$):
 - $C \models [a]^k \langle a \rangle tt$ for all $k \in \mathbb{N}$
 - $D_n \models [a]^k \langle a \rangle tt$ for all $0 \leq k \leq n$
 - $D_n \not\models [a]^k \langle a \rangle tt$ for all $k > n$
- Conclusion: no single HML formula can distinguish C and all D_n
- Generally: **invariant** property “always $\langle a \rangle tt$ ” not expressible
- Requires **infinite conjunction**:

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

Infinite Disjunctions

Dually: **possibility** properties expressible by infinite disjunctions

Example 4.3

- Let $C = a.C$, $D = a.D + a.nil$ as before
- C has no **possibility** to terminate
- D has the option to terminate (i.e., to eventually satisfy $[a]ff$) at any time by choosing the $a.nil$ branch

Infinite Disjunctions

Dually: **possibility** properties expressible by infinite disjunctions

Example 4.3

- Let $C = a.C$, $D = a.D + a.nil$ as before
- C has no **possibility** to terminate
- D has the option to terminate (i.e., to eventually satisfy $[a]ff$) at any time by choosing the $a.nil$ branch
- Representable by **infinite disjunction**:

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

Adding Recursion to HML

Infinite Disjunctions

Dually: **possibility** properties expressible by infinite disjunctions

Example 4.3

- Let $C = a.C$, $D = a.D + a.nil$ as before
- C has no **possibility** to terminate
- D has the option to terminate (i.e., to eventually satisfy $[a]ff$) at any time by choosing the $a.nil$ branch
- Representable by **infinite disjunction**:

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

Problem: infinite formulae not easy to handle

Introducing Recursion

Solution: employ recursion!

- $Inv(\langle a \rangle tt) \equiv \langle a \rangle tt \wedge [a] Inv(\langle a \rangle tt)$
- $Pos([a]ff) \equiv [a]ff \vee \langle a \rangle Pos([a]ff)$

Introducing Recursion

Solution: employ recursion!

- $Inv(\langle a \rangle tt) \equiv \langle a \rangle tt \wedge [a] Inv(\langle a \rangle tt)$
- $Pos([a]ff) \equiv [a]ff \vee \langle a \rangle Pos([a]ff)$

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

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

Adding Recursion to HML

Introducing Recursion

Solution: employ recursion!

- $Inv(\langle a \rangle tt) \equiv \langle a \rangle tt \wedge [a] Inv(\langle a \rangle tt)$
- $Pos([a] ff) \equiv [a] ff \vee \langle a \rangle Pos([a] ff)$

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

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

Open questions

- Do such recursive equations (always) have **solutions**?
- If so, are they **unique**?
- How can we **compute** whether a process satisfies a recursive formula?

Existence of Solutions

Example 4.4

- Consider again $C = a.C$, $D = a.D + a.nil$

Existence of Solutions

Example 4.4

- Consider again $C = a.C$, $D = a.D + a.nil$
 - Invariant: $X \equiv \langle a \rangle tt \wedge [a]X$
 - $X = \emptyset$ is a solution (as no process can satisfy both $\langle a \rangle tt$ and $[a]ff$)
 - but we expect $C \in X$ (as C can perform a invariantly)
 - in fact, $X = \{C\}$ also solves the equation (and is the **greatest solution** w.r.t. \subseteq)
- \implies write $X \stackrel{max}{=} \langle a \rangle tt \wedge [a]X$

Adding Recursion to HML

Existence of Solutions

Example 4.4

- Consider again $C = a.C$, $D = a.D + a.nil$
 - Invariant: $X \equiv \langle a \rangle tt \wedge [a]X$
 - $X = \emptyset$ is a solution (as no process can satisfy both $\langle a \rangle tt$ and $[a]ff$)
 - but we expect $C \in X$ (as C can perform a invariantly)
 - in fact, $X = \{C\}$ also solves the equation (and is the **greatest solution** w.r.t. \subseteq)
- \implies write $X \stackrel{max}{=} \langle a \rangle tt \wedge [a]X$
- Possibility: $Y \equiv [a]ff \vee \langle a \rangle Y$
 - greatest solution: $Y = \{C, D, nil\}$
 - but we expect $C \notin Y$ (as C cannot terminate at all)
 - here: **least solution** w.r.t. \subseteq : $Y = \{D, nil\}$
- \implies write $Y \stackrel{min}{=} [a]ff \vee \langle a \rangle Y$

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 4.5

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

- **Invariant:** $Inv(F) \equiv X$ for $X \stackrel{max}{=} F \wedge [Act]X$
 - $s \models Inv(F)$ if all states reachable from s satisfy F

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 4.5

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

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

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 4.5

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

- **Invariant:** $Inv(F) \equiv X$ for $X \stackrel{max}{=} F \wedge [Act]X$
 - $s \models Inv(F)$ if all states reachable from s satisfy F
- **Possibility:** $Pos(F) \equiv Y$ for $Y \stackrel{min}{=} F \vee \langle Act \rangle Y$
 - $s \models Pos(F)$ if a state satisfying F is reachable from s
- **Safety:** $Safe(F) \equiv X$ for $X \stackrel{max}{=} F \wedge ([Act]ff \vee \langle Act \rangle X)$
 - $s \models Safe(F)$ if s has a complete (i.e., infinite or terminating) transition sequence where each state satisfies F

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 4.5

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

- **Invariant:** $Inv(F) \equiv X$ for $X \stackrel{max}{=} F \wedge [Act]X$
 - $s \models Inv(F)$ if all states reachable from s satisfy F
- **Possibility:** $Pos(F) \equiv Y$ for $Y \stackrel{min}{=} F \vee \langle Act \rangle Y$
 - $s \models Pos(F)$ if a state satisfying F is reachable from s
- **Safety:** $Safe(F) \equiv X$ for $X \stackrel{max}{=} F \wedge ([Act]ff \vee \langle Act \rangle X)$
 - $s \models Safe(F)$ if s has a complete (i.e., infinite or terminating) transition sequence where each state satisfies F
- **Eventuality:** $Evt(F) \equiv Y$ for $Y \stackrel{min}{=} F \vee (\langle Act \rangle tt \wedge [Act]Y)$
 - $s \models Evt(F)$ if each complete transition sequence starting in s contains a state satisfying F

HML with One Recursive Variable

Outline of Lecture 4

Recap: Hennessy-Milner Logic

Adding Recursion to HML

HML with One Recursive Variable

Complete Lattices

HML with One Recursive Variable

Syntax of HML with One Recursive Variable

Initially: only **one variable**

Later: **mutual recursion**

HML with One Recursive Variable

Syntax of HML with One Recursive Variable

Initially: only **one variable**

Later: **mutual recursion**

Definition 4.6 (Syntax of HML with one variable)

The set HMF_X of **Hennessy-Milner formulae with one variable X** over a set of actions Act is defined by the following syntax:

$F ::= X$	(variable)
tt	(true)
ff	(false)
$F_1 \wedge F_2$	(conjunction)
$F_1 \vee F_2$	(disjunction)
$\langle \alpha \rangle F$	(diamond)
$[\alpha] F$	(box)

where $\alpha \in Act$.

HML with One Recursive Variable

Semantics of HML with One Recursive Variable I

So far: $\llbracket F \rrbracket \subseteq S$ for $F \in HMF$ and LTS $(S, Act, \longrightarrow)$

Now: semantics of formula depends on states that (are assumed to) satisfy X

HML with One Recursive Variable

Semantics of HML with One Recursive Variable I

So far: $\llbracket F \rrbracket \subseteq S$ for $F \in HMF$ and LTS $(S, Act, \longrightarrow)$

Now: semantics of formula depends on states that (are assumed to) satisfy X

Definition 4.7 (Semantics of HML with one variable)

Let $(S, Act, \longrightarrow)$ be an LTS and $F \in HMF_X$. The **semantics** of F ,

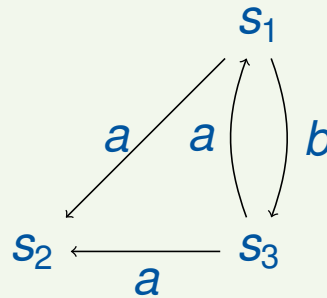
$$\llbracket F \rrbracket : 2^S \rightarrow 2^S,$$

is defined by

$$\begin{aligned}\llbracket X \rrbracket (T) &:= T \\ \llbracket \text{tt} \rrbracket (T) &:= S \\ \llbracket \text{ff} \rrbracket (T) &:= \emptyset \\ \llbracket F_1 \wedge F_2 \rrbracket (T) &:= \llbracket F_1 \rrbracket (T) \cap \llbracket F_2 \rrbracket (T) \\ \llbracket F_1 \vee F_2 \rrbracket (T) &:= \llbracket F_1 \rrbracket (T) \cup \llbracket F_2 \rrbracket (T) \\ \llbracket \langle \alpha \rangle F \rrbracket (T) &:= \langle \cdot \alpha \cdot \rangle (\llbracket F \rrbracket (T)) \\ \llbracket [\alpha] F \rrbracket (T) &:= [\cdot \alpha \cdot] (\llbracket F \rrbracket (T))\end{aligned}$$

Semantics of HML with One Recursive Variable II

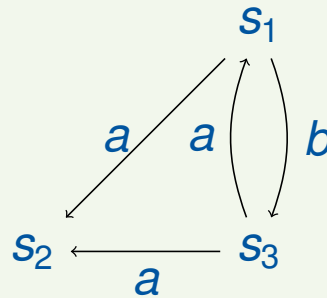
Example 4.8



Let $S := \{s_1, s_2, s_3\}$.

Semantics of HML with One Recursive Variable II

Example 4.8

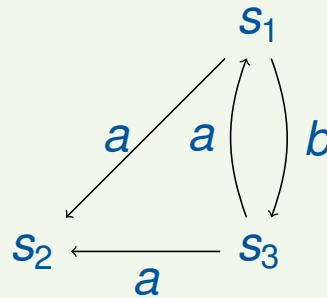


Let $S := \{s_1, s_2, s_3\}$.

- $\llbracket \langle a \rangle X \rrbracket (\{s_1\}) = \{s_3\}$

Semantics of HML with One Recursive Variable II

Example 4.8

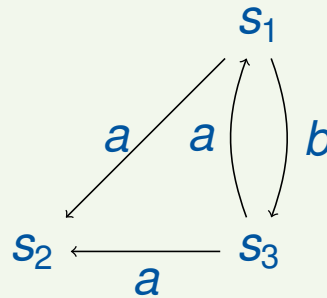


Let $S := \{s_1, s_2, s_3\}$.

- $\llbracket \langle a \rangle X \rrbracket (\{s_1\}) = \{s_3\}$
- $\llbracket \langle a \rangle X \rrbracket (\{s_1, s_2\}) = \{s_1, s_3\}$

Semantics of HML with One Recursive Variable II

Example 4.8



Let $S := \{s_1, s_2, s_3\}$.

- $\llbracket \langle a \rangle X \rrbracket (\{s_1\}) = \{s_3\}$
- $\llbracket \langle a \rangle X \rrbracket (\{s_1, s_2\}) = \{s_1, s_3\}$
- $\llbracket \langle b \rangle X \rrbracket (\{s_2\}) = \{s_2, s_3\}$

Semantics of HML with One Recursive Variable III

- Idea underlying the definition of

$$\llbracket \cdot \rrbracket : HMF_X \rightarrow (2^S \rightarrow 2^S) :$$

if $T \subseteq S$ gives the set of states that satisfy X , then $\llbracket F \rrbracket(T)$ will be the set of states that satisfy F

Semantics of HML with One Recursive Variable III

- Idea underlying the definition of

$$[[\cdot]] : HMF_X \rightarrow (2^S \rightarrow 2^S) :$$

if $T \subseteq S$ gives the set of states that satisfy X , then $[[F]](T)$ will be the set of states that satisfy F

- How to determine this T ?
- According to previous discussion: as solution of **recursive equation** of the form $X = F_X$ where $F_X \in HMF_X$

HML with One Recursive Variable

Semantics of HML with One Recursive Variable III

- Idea underlying the definition of

$$\llbracket \cdot \rrbracket : HMF_X \rightarrow (2^S \rightarrow 2^S) :$$

if $T \subseteq S$ gives the set of states that satisfy X , then $\llbracket F \rrbracket(T)$ will be the set of states that satisfy F

- How to determine this T ?
- According to previous discussion: as solution of **recursive equation** of the form $X = F_X$ where $F_X \in HMF_X$
- But: solution **not unique**; therefore write:

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

HML with One Recursive Variable

Semantics of HML with One Recursive Variable III

- Idea underlying the definition of

$$\llbracket \cdot \rrbracket : HMF_X \rightarrow (2^S \rightarrow 2^S) :$$

if $T \subseteq S$ gives the set of states that satisfy X , then $\llbracket F \rrbracket(T)$ will be the set of states that satisfy F

- How to determine this T ?
- According to previous discussion: as solution of **recursive equation** of the form $X = F_X$ where $F_X \in HMF_X$
- But: solution **not unique**; therefore write:

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

- In the following we will see:
 1. Equation $X = F_X$ always **solvable**
 2. Least and greatest solutions are **unique** and can be obtained by **fixed-point iteration**

Complete Lattices

Outline of Lecture 4

Recap: Hennessy-Milner Logic

Adding Recursion to HML

HML with One Recursive Variable

Complete Lattices

Complete Lattices

Partial Orders

Definition 4.9 (Partial order)

A **partial order (PO)** (D, \sqsubseteq) consists of a set D , called **domain**, and of a relation $\sqsubseteq \subseteq D \times D$ such that, for every $d_1, d_2, d_3 \in D$,

reflexivity: $d_1 \sqsubseteq d_1$

transitivity: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

antisymmetry: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always $d_1 \sqsubseteq d_2$ or $d_2 \sqsubseteq d_1$.

Complete Lattices

Partial Orders

Definition 4.9 (Partial order)

A **partial order (PO)** (D, \sqsubseteq) consists of a set D , called **domain**, and of a relation $\sqsubseteq \subseteq D \times D$ such that, for every $d_1, d_2, d_3 \in D$,

reflexivity: $d_1 \sqsubseteq d_1$

transitivity: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

antisymmetry: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always $d_1 \sqsubseteq d_2$ or $d_2 \sqsubseteq d_1$.

Example 4.10

1. (\mathbb{N}, \leq) is a total partial order

Complete Lattices

Partial Orders

Definition 4.9 (Partial order)

A **partial order (PO)** (D, \sqsubseteq) consists of a set D , called **domain**, and of a relation $\sqsubseteq \subseteq D \times D$ such that, for every $d_1, d_2, d_3 \in D$,

reflexivity: $d_1 \sqsubseteq d_1$

transitivity: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

antisymmetry: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always $d_1 \sqsubseteq d_2$ or $d_2 \sqsubseteq d_1$.

Example 4.10

1. (\mathbb{N}, \leq) is a total partial order
2. $(\mathbb{N}, <)$ is not a partial order (since not reflexive)

Complete Lattices

Partial Orders

Definition 4.9 (Partial order)

A **partial order (PO)** (D, \sqsubseteq) consists of a set D , called **domain**, and of a relation $\sqsubseteq \subseteq D \times D$ such that, for every $d_1, d_2, d_3 \in D$,

reflexivity: $d_1 \sqsubseteq d_1$

transitivity: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

antisymmetry: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always $d_1 \sqsubseteq d_2$ or $d_2 \sqsubseteq d_1$.

Example 4.10

1. (\mathbb{N}, \leq) is a total partial order
2. $(\mathbb{N}, <)$ is not a partial order (since not reflexive)
3. $(2^{\mathbb{N}}, \subseteq)$ is a (non-total) partial order

Complete Lattices

Partial Orders

Definition 4.9 (Partial order)

A **partial order (PO)** (D, \sqsubseteq) consists of a set D , called **domain**, and of a relation $\sqsubseteq \subseteq D \times D$ such that, for every $d_1, d_2, d_3 \in D$,

reflexivity: $d_1 \sqsubseteq d_1$

transitivity: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

antisymmetry: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always $d_1 \sqsubseteq d_2$ or $d_2 \sqsubseteq d_1$.

Example 4.10

1. (\mathbb{N}, \leq) is a total partial order
2. $(\mathbb{N}, <)$ is not a partial order (since not reflexive)
3. $(2^{\mathbb{N}}, \subseteq)$ is a (non-total) partial order
4. (Σ^*, \sqsubseteq) is a (non-total) partial order, where Σ is some alphabet and \sqsubseteq denotes prefix ordering ($u \sqsubseteq v \iff \exists w \in \Sigma^* : uw = v$)

Complete Lattices

Upper and Lower Bounds

Definition 4.11 ((Least) upper bounds and (greatest) lower bounds)

Let (D, \sqsubseteq) be a partial order and $T \subseteq D$.

1. An element $d \in D$ is called an **upper bound** of T if $t \sqsubseteq d$ for every $t \in T$ (notation: $T \sqsubseteq d$). It is called **least upper bound (LUB)** (or **supremum**) of T if additionally $d \sqsubseteq d'$ for every upper bound d' of T (notation: $d = \bigsqcup T$).

Complete Lattices

Upper and Lower Bounds

Definition 4.11 ((Least) upper bounds and (greatest) lower bounds)

Let (D, \sqsubseteq) be a partial order and $T \subseteq D$.

1. An element $d \in D$ is called an **upper bound** of T if $t \sqsubseteq d$ for every $t \in T$ (notation: $T \sqsubseteq d$). It is called **least upper bound (LUB)** (or **supremum**) of T if additionally $d \sqsubseteq d'$ for every upper bound d' of T (notation: $d = \bigsqcup T$).
2. An element $d \in D$ is called an **lower bound** of T if $d \sqsubseteq t$ for every $t \in T$ (notation: $d \sqsubseteq T$). It is called **greatest lower bound (GLB)** (or **infimum**) of T if $d' \sqsubseteq d$ for every lower bound d' of T (notation: $d = \bigsqcap T$).

Complete Lattices

Upper and Lower Bounds

Definition 4.11 ((Least) upper bounds and (greatest) lower bounds)

Let (D, \sqsubseteq) be a partial order and $T \subseteq D$.

1. An element $d \in D$ is called an **upper bound** of T if $t \sqsubseteq d$ for every $t \in T$ (notation: $T \sqsubseteq d$). It is called **least upper bound (LUB)** (or **supremum**) of T if additionally $d \sqsubseteq d'$ for every upper bound d' of T (notation: $d = \bigsqcup T$).
2. An element $d \in D$ is called an **lower bound** of T if $d \sqsubseteq t$ for every $t \in T$ (notation: $d \sqsubseteq T$). It is called **greatest lower bound (GLB)** (or **infimum**) of T if $d' \sqsubseteq d$ for every lower bound d' of T (notation: $d = \bigsqcap T$).

Example 4.12

1. $T \subseteq \mathbb{N}$ has a LUB/GLB in (\mathbb{N}, \leq) iff it is finite/non-empty

Complete Lattices

Upper and Lower Bounds

Definition 4.11 ((Least) upper bounds and (greatest) lower bounds)

Let (D, \sqsubseteq) be a partial order and $T \subseteq D$.

1. An element $d \in D$ is called an **upper bound** of T if $t \sqsubseteq d$ for every $t \in T$ (notation: $T \sqsubseteq d$). It is called **least upper bound (LUB)** (or **supremum**) of T if additionally $d \sqsubseteq d'$ for every upper bound d' of T (notation: $d = \bigsqcup T$).
2. An element $d \in D$ is called an **lower bound** of T if $d \sqsubseteq t$ for every $t \in T$ (notation: $d \sqsubseteq T$). It is called **greatest lower bound (GLB)** (or **infimum**) of T if $d' \sqsubseteq d$ for every lower bound d' of T (notation: $d = \bigsqcap T$).

Example 4.12

1. $T \subseteq \mathbb{N}$ has a LUB/GLB in (\mathbb{N}, \leq) iff it is finite/non-empty
2. In $(2^{\mathbb{N}}, \subseteq)$, every subset $T \subseteq 2^{\mathbb{N}}$ has an LUB and GLB:

$$\bigsqcup T = \bigcup T \quad \text{and} \quad \bigsqcap T = \bigcap T$$

Complete Lattices

Complete Lattices

Definition 4.13 (Complete lattice)

A **complete lattice** is a partial order (D, \sqsubseteq) such that all subsets of D have LUBs and GLBs. In this case,

$$\perp := \bigsqcup \emptyset (= \bigsqcap D) \quad \text{and} \quad \top := \bigsqcap \emptyset (= \bigsqcup D)$$

respectively denote the **least and greatest element** of D .

Complete Lattices

Complete Lattices

Definition 4.13 (Complete lattice)

A **complete lattice** is a partial order (D, \sqsubseteq) such that all subsets of D have LUBs and GLBs. In this case,

$$\perp := \bigsqcup \emptyset (= \bigsqcap D) \quad \text{and} \quad \top := \bigsqcap \emptyset (= \bigsqcup D)$$

respectively denote the **least and greatest element** of D .

Example 4.14

1. (\mathbb{N}, \leq) is not a complete lattice as, e.g., \mathbb{N} does not have a LUB

Complete Lattices

Complete Lattices

Definition 4.13 (Complete lattice)

A **complete lattice** is a partial order (D, \sqsubseteq) such that all subsets of D have LUBs and GLBs. In this case,

$$\perp := \bigsqcup \emptyset (= \bigsqcap D) \quad \text{and} \quad \top := \bigsqcap \emptyset (= \bigsqcup D)$$

respectively denote the **least and greatest element** of D .

Example 4.14

1. (\mathbb{N}, \leq) is not a complete lattice as, e.g., \mathbb{N} does not have a LUB
2. $(\mathbb{N} \cup \{\infty\}, \leq)$ with $n \leq \infty$ for all $n \in \mathbb{N}$ is a complete lattice

Complete Lattices

Complete Lattices

Definition 4.13 (Complete lattice)

A **complete lattice** is a partial order (D, \sqsubseteq) such that all subsets of D have LUBs and GLBs. In this case,

$$\perp := \bigsqcup \emptyset (= \bigsqcap D) \quad \text{and} \quad \top := \bigsqcap \emptyset (= \bigsqcup D)$$

respectively denote the **least and greatest element** of D .

Example 4.14

1. (\mathbb{N}, \leq) is not a complete lattice as, e.g., \mathbb{N} does not have a LUB
2. $(\mathbb{N} \cup \{\infty\}, \leq)$ with $n \leq \infty$ for all $n \in \mathbb{N}$ is a complete lattice
3. $(2^{\mathbb{N}}, \subseteq)$ is a complete lattice

Complete Lattices

Application to HML with Recursion

Lemma 4.15

Let $(S, Act, \longrightarrow)$ be an LTS. Then $(2^S, \subseteq)$ is a complete lattice with

- $\bigsqcup \mathcal{T} = \bigcup \mathcal{T} = \bigcup_{T \in \mathcal{T}} T$ for all $\mathcal{T} \subseteq 2^S$
- $\bigsqcap \mathcal{T} = \bigcap \mathcal{T} = \bigcap_{T \in \mathcal{T}} T$ for all $\mathcal{T} \subseteq 2^S$

Complete Lattices

Application to HML with Recursion

Lemma 4.15

Let $(S, Act, \longrightarrow)$ be an LTS. Then $(2^S, \subseteq)$ is a complete lattice with

- $\bigsqcup \mathcal{T} = \bigcup \mathcal{T} = \bigcup_{T \in \mathcal{T}} T$ for all $\mathcal{T} \subseteq 2^S$
- $\bigsqcap \mathcal{T} = \bigcap \mathcal{T} = \bigcap_{T \in \mathcal{T}} T$ for all $\mathcal{T} \subseteq 2^S$
- $\perp = \bigsqcup \emptyset = \bigsqcap 2^S = \emptyset$
- $\top = \bigsqcap \emptyset = \bigsqcup 2^S = S$

Complete Lattices

Application to HML with Recursion

Lemma 4.15

Let $(S, Act, \longrightarrow)$ be an LTS. Then $(2^S, \subseteq)$ is a complete lattice with

- $\bigsqcup \mathcal{T} = \bigcup \mathcal{T} = \bigcup_{T \in \mathcal{T}} T$ for all $\mathcal{T} \subseteq 2^S$
- $\bigsqcap \mathcal{T} = \bigcap \mathcal{T} = \bigcap_{T \in \mathcal{T}} T$ for all $\mathcal{T} \subseteq 2^S$
- $\perp = \bigsqcup \emptyset = \bigsqcap 2^S = \emptyset$
- $\top = \bigsqcap \emptyset = \bigsqcup 2^S = S$

Proof.

omitted □