Recording agreement

- The lectures will be recorded.
- You are not obligated neither to enable video or audio nor to share other personal data as real names or photos of yourself.
- The transmission of video and audio is disabled by default when joining the lecture. If you do not want to be part of the recording, simply do not activate the transmission.

- If you agree to be part of the recording, read the conditions provided in Moodle. When activating video or audio, you agree with the terms.

- Note that the breakout rooms are not recorded.

Satisfiability Checking
02 Propositional logic I

Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 21/22

Abstract syntax of well-formed propositional formulae:

$$\varphi \;:=\; a \;\mid\; (\neg\varphi) \;\mid\; (\varphi \wedge \varphi)$$

where AP is a set of (atomic) propositions (Boolean variables) and $a \in AP$.
We write *PropForm* for the set of all propositional logic formulae.

Syntactic sugar:

$$\begin{array}{rcl}
\bot & := & (a \wedge \neg a) \\
\top & := & (a \vee \neg a) \\
(\varphi_1 \vee \varphi_2) & := & \neg((\neg\varphi_1) \wedge (\neg\varphi_2)) \\
(\varphi_1 \rightarrow \varphi_2) & := & ((\neg\varphi_1) \vee \varphi_2) \\
(\varphi_1 \leftrightarrow \varphi_2) & := & ((\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)) \\
(\varphi_1 \bigoplus \varphi_2) & := & (\varphi_1 \leftrightarrow (\neg\varphi_2))
\end{array}$$

Formulae

- Examples of well-formed formulae:
  - $(\neg a)$
  - $(\neg(\neg a))$
  - $(a \land (b \land c))$
  - $(a \rightarrow (b \rightarrow c))$
- We omit parentheses whenever we may restore them through operator precedence:

  binds stronger

  $$\xleftarrow{\qquad\qquad\qquad\qquad}$$

  $$\neg \quad \land \quad \lor \quad \rightarrow \quad \leftrightarrow$$

- We will also use the "big" Boolean notation, e.g.

$$\bigwedge_{i=1}^{5} x_i \qquad \textit{is defined as} \qquad x_1 \land x_2 \land x_3 \land x_4 \land x_5 \;.$$

Semantics: Assignments

Structures for predicate logic:

- The domain is $\mathbb{B} = \{0, 1\}$.
- The interpretation assigns Boolean values to the variables:

$$\alpha : AP \rightarrow \{0, 1\}$$

We call these special interpretations assignments and use *Assign* to denote the set of all assignments.
Example: $AP = \{a, b\}, \alpha(a) = 0, \alpha(b) = 1$

Equivalently, we can see an assignment α as a set of variables ($\alpha \in 2^{AP}$), defining the variables from the set to be true and the others false.
Example: $AP = \{a, b\}, \alpha = \{b\}$

An assignment can also be seen as being of type $\alpha \in \{0, 1\}^{AP}$, if we have an order on the propositions.
Example: $AP = \{a, b\}, \alpha = 01$

Only the projected assignment matters...

- Let $\alpha_1, \alpha_2 \in \mathit{Assign}$ and $\varphi \in \mathit{PropForm}$.
- Let $AP(\varphi)$ be the atomic propositions in φ.
- Clearly $AP(\varphi) \subseteq AP$.
- Lemma: if $\alpha_1|_{AP(\varphi)} = \alpha_2|_{AP(\varphi)}$, then

Projection

$$(\alpha_1 \text{ satisfies } \varphi) \quad \text{iff} \quad (\alpha_2 \text{ satisfies } \varphi)$$

- We will assume, for simplicity, that $AP = AP(\varphi)$.

Semantics I: Truth tables

- Truth tables define the semantics (=meaning) of the operators. They can be used to define the semantics of formulae inductively over their structure.
- Convention: 0= false, 1= true

a⊕b = (¬a ∧ b) ∨ (a ∧¬b

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$p \bigoplus q$
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	0	0	1
1	1	0	1	1	1	1	0

Each possible assignment is covered by a line of the truth table.
α satisfies φ iff in the line for α and the column for φ the entry is 1.

Q: How many binary operators can we define that have different semantics?
A: 16

Semantics I: Example

- Let φ be defined as $(a \vee (b \rightarrow c))$.
- Let $\alpha : \{a, b, c\} \rightarrow \{0, 1\}$ be an assignment with $\alpha(a) = 0$, $\alpha(b) = 0$, and $\alpha(c) = 1$.
- Q: Does α satisfy φ?
- A1: Compute with truth table:

a	b	c	$b \rightarrow c$	$a \vee (b \rightarrow c)$
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	1
1	1	1	1	1

Semantics II: Satisfaction relation

满足符

Satisfaction relation: $\models \subseteq \textit{Assign} \times \textit{PropForm}$

Instead of $(\alpha, \varphi) \in \models$ we write $\alpha \models \varphi$ and say that

- α satisfies φ or
- φ holds for α or
- α is a model of φ.

$\models$ is defined recursively:

$$
\begin{array}{lll}
\alpha \models p & \textit{iff} & \alpha(p) = \textit{true} \\
\alpha \models \neg\varphi & \textit{iff} & \alpha \not\models \varphi \\
\alpha \models \varphi_1 \wedge \varphi_2 & \textit{iff} & \alpha \models \varphi_1 \textit{ and } \alpha \models \varphi_2 \\
\alpha \models \varphi_1 \vee \varphi_2 & \textit{iff} & \alpha \models \varphi_1 \textit{ or } \alpha \models \varphi_2 \\
\alpha \models \varphi_1 \rightarrow \varphi_2 & \textit{iff} & \alpha \models \varphi_1 \textit{ implies } \alpha \models \varphi_2 \\
\alpha \models \varphi_1 \leftrightarrow \varphi_2 & \textit{iff} & \alpha \models \varphi_1 \textit{ iff } \alpha \models \varphi_2
\end{array}
$$

Note: More elegant but semantically equivalent to truth tables.

Semantics II: Example

- Let φ be defined as $(a \vee (b \rightarrow c))$.
- Let $\alpha : \{a, b, c\} \rightarrow \{0, 1\}$ be an assignment with $\alpha(a) = 0$, $\alpha(b) = 0$, and $\alpha(c) = 1$.

- Q: Does α satisfy φ?

  A2: Compute with the satisfaction relation:

$$
\begin{array}{ll}
 & \alpha \models (a \vee (b \rightarrow c)) \\
\textit{iff} & \alpha \models a \textit{ or } \alpha \models (b \rightarrow c) \\
\textit{iff} & \alpha \models a \textit{ or } (\alpha \models b \textit{ implies } \alpha \models c) \\
\textit{iff} & 0 \textit{ or } (0 \textit{ implies } 1) \\
\textit{iff} & 0 \textit{ or } 1 \\
\textit{iff} & 1
\end{array}
$$

- Using the satisfaction relation we can define an algorithm for the problem to decide whether an assignment $\alpha : AP \to \{0,1\}$ is a model of a propositional logic formula $\varphi \in \mathit{PropForm}$:

```
Eval(α, φ) {
   if φ ≡ a return α(a);
   if φ ≡ (¬φ1) return not Eval(α, φ1);
   if φ ≡ (φ1 op φ2)
         return Eval(α, φ1) ⟦op⟧ Eval(α, φ2);
}
```

- Equivalent to the $\models$ relation, but from the algorithmic view.
- Q: Complexity? A: Polynomial (time and space).

Semantics III: Example

- Recall our example
  - $\varphi = (a \vee (b \to c))$
  - $\alpha : \{a, b, c\} \to \{0, 1\}$ with $\alpha(a) = 0$, $\alpha(b) = 0$, and $\alpha(c) = 1$.

- $$\begin{aligned} \mathit{Eval}(\alpha,\, \varphi) \;=\; & \mathit{Eval}(\alpha,\, a) \textit{ or } \mathit{Eval}(\alpha,\, b \to c) = \\ & 0 \textit{ or } (\mathrm{Eval}(\alpha,\, b) \textit{ implies } \mathrm{Eval}(\alpha,\, c)) = \\ & 0 \textit{ or } (0 \textit{ implies } 1) = \\ & 0 \textit{ or } 1 = \\ & 1 \end{aligned}$$

- Hence, $\alpha \models \varphi$.

Satisfying assignments

- Intuition: each formula specifies a set of assignments satisfying it.
- Remember: *Assign* denotes the set of all assignments.
- Function $\mathit{sat} : \mathit{PropForm} \to 2^{\mathit{Assign}}$

  (a formula $\to$ set of its satisfying assignments)
- Recursive definition:

$$\begin{array}{lcl}
\mathit{sat}(a) & = & \{\alpha \mid \alpha(a) = 1\}, \quad a \in AP \\
\mathit{sat}(\neg\varphi_1) & = & \mathit{Assign} \setminus \mathit{sat}(\varphi_1) \\
\mathit{sat}(\varphi_1 \wedge \varphi_2) & = & \mathit{sat}(\varphi_1) \cap \mathit{sat}(\varphi_2) \\
\mathit{sat}(\varphi_1 \vee \varphi_2) & = & \mathit{sat}(\varphi_1) \cup \mathit{sat}(\varphi_2) \\
\mathit{sat}(\varphi_1 \to \varphi_2) & = & (\mathit{Assign} \setminus \mathit{sat}(\varphi_1)) \cup \mathit{sat}(\varphi_2)
\end{array}$$

- For $\varphi \in \mathit{PropForm}$ and $\alpha \in \mathit{Assign}$ it holds that

$$\alpha \models \varphi \qquad \mathit{iff} \qquad \alpha \in \mathit{sat}(\varphi)$$

Satisfying assignments: Example

$$
\begin{aligned}
& sat(a \vee (b \to c)) && = \\
& sat(a) \cup sat(b \to c) && = \\
& sat(a) \cup ((Assign \setminus sat(b)) \cup sat(c)) && = \\
& \{\alpha \in Assign \mid \alpha(a) = 1\} \cup \\
& \{\alpha \in Assign \mid \alpha(b) = 0\} \cup \\
& \{\alpha \in Assign \mid \alpha(c) = 1\} && = \\
& \{\alpha \in Assign \mid \alpha(a) = 1 \text{ or } \alpha(b) = 0 \text{ or } \alpha(c) = 1\}
\end{aligned}
$$

Extensions of $\models$

- We define $\models\ \subseteq\ 2^{Assign} \times PropForm$ by

$$T \models \varphi \text{ iff } T \subseteq sat(\varphi)$$

for formulae $\varphi \in PropForm$ and assignment sets $T \subseteq 2^{Assign}$.

Examples: $\{\alpha \in Assign \mid \alpha(a) = \alpha(c) = 1\} \models a \lor (b \rightarrow c)$
$\{\alpha \in Assign \mid \alpha(x_1) = 1\} \models x_1 \lor x_2$

- We define $\models\ \subseteq\ PropForm \times PropForm$ by

$$\varphi_1 \models \varphi_2 \text{ iff } sat(\varphi_1) \subseteq sat(\varphi_2)$$

for formulae $\varphi_1, \varphi_2 \in PropForm$.

Examples: $a \land c \models a \lor (b \rightarrow c)$
$x_1 \models x_1 \lor x_2$

Short summary for propositional logic syntax and semantics

- Syntax of propositional formulae $\varphi \in \mathit{PropForm}$:

$$\varphi \ := \ \mathit{AP} \mid (\neg\varphi) \mid (\varphi \land \varphi)$$

- Semantics:
  - Assignments $\alpha \in \mathit{Assign}$:

$$\begin{array}{l} \alpha : \mathit{AP} \to \{0,1\} \\ \alpha \in 2^{\mathit{AP}} \\ \alpha \in \{0,1\}^{\mathit{AP}} \end{array}$$

  - Satisfaction relation:

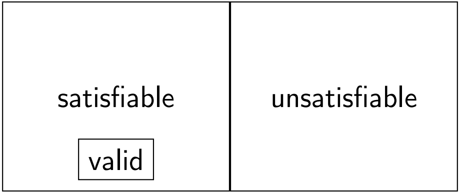
$$\begin{array}{lll} \models \ \subseteq \mathit{Assign} \times \mathit{PropForm} & , & (\text{e.g., } \alpha \models \varphi \ ) \\ \models \ \subseteq 2^{\mathit{Assign}} \times \mathit{PropForm} & , & (\text{e.g., } \{\alpha_1, \ldots, \alpha_n\} \models \varphi \ ) \\ \models \ \subseteq \mathit{PropForm} \times \mathit{PropForm}, & & (\text{e.g., } \varphi_1 \models \varphi_2) \\ \mathit{sat} : \ \mathit{PropForm} \to 2^{\mathit{Assign}} & , & (\text{e.g., } \ \mathit{sat}(\varphi) \ ) \end{array}$$

Semantic classification of formulae

- A formula φ is called valid if $sat(\varphi) = Assign$.
  (Also called a tautology).

- A formula φ is called satisfiable if $sat(\varphi) \neq \emptyset$.

- A formula φ is called unsatisfiable if $sat(\varphi) = \emptyset$.
  (Also called a contradiction).



Some notations

- We can write:
  - $\models \varphi$ when φ is valid
  - $\not\models \varphi$ when φ is not valid
  - $\not\models \neg\varphi$ when φ is satisfiable
  - $\models \neg\varphi$ when φ is unsatisfiable

Examples

- $(x_1 \wedge x_2) \to (x_1 \vee x_2)$ is valid
- $(x_1 \vee x_2) \to x_1$ is satisfiable
- $(x_1 \wedge x_2) \wedge \neg x_1$ is unsatisfiable

Examples

- Here are some valid formulae:
  - $\models a \wedge 1 \leftrightarrow a$
  - $\models a \wedge 0 \leftrightarrow 0$
  - $\models \neg\neg a \leftrightarrow a$ (double-negation rule)
  - $\models a \wedge (b \vee c) \leftrightarrow (a \wedge b) \vee (a \wedge c)$

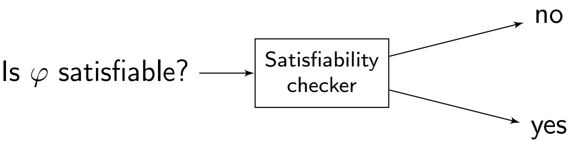
- Some more (De Morgan rules):
  - $\models \neg(a \wedge b) \leftrightarrow (\neg a \vee \neg b)$
  - $\models \neg(a \vee b) \leftrightarrow (\neg a \wedge \neg b)$

The satisfiability problem for propositional logic

- The satisfiability problem for propositional logic is as follows:

  Given an input propositional formula φ, decide whether φ is satisfiable.
- This problem is decidable but NP-complete.
- An algorithm that always terminates for each propositional logic formula with the correct answer is called a decision procedure for propositional logic.

Goal: Design and implement such a decision procedure:



Note: A formula φ is valid iff $\neg\varphi$ is unsatisfiable.

Learning target

- What are the rules to (syntactically) build propositional logic formulas?
- How to interpret propositional logic formulas...
  - ... using truth tables?
  - ... using the satisfaction relation?
  - ... algorithmically by the `Eval` function?
- How to compute the set of all satisfying assignments recursively by the `sat` function?
- When is a propositional logic formula valid, satisfiable or unsatisfiable?