Satisfiability Checking

02 Propositional logic I

Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 23/24

Syntax of propositional logic

Abstract syntax of well-formed propositional formulae:

A proposition is a sentence like p→(p→q)

$$\varphi \ := \ a \ | \ (\neg\varphi) \ | \ (\varphi \land \varphi)$$

. An atomic proposition is like p

原子命题是最小的命题, 内部不包含更小的命题

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.

formulae由 ap构成

Syntactic sugar:

$$\boxed{\bot \quad := (a \land \neg a)}$$ unconditional false

$$\boxed{\top \quad := (a \lor \neg a)}$$ unconditional true

$$(\varphi_1 \lor \varphi_2) := \neg((\neg\varphi_1) \land (\neg\varphi_2))$$

$$(\varphi_1 \to \varphi_2) := ((\neg\varphi_1) \lor \varphi_2)$$ ¬(ϕ1^(¬ϕ2))

$$(\varphi_1 \leftrightarrow \varphi_2) := ((\varphi_1 \to \varphi_2) \land (\varphi_2 \to \varphi_1))$$

$$(\varphi_1 \oplus \varphi_2) := (\varphi_1 \leftrightarrow (\neg\varphi_2))$$

Formulae

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

  binds stronger

  $\longleftarrow$

  $\neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow$

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

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

Semantics: Assignments

α 为assignment 函数, 对atomatic proposition进行true/false(0/1)赋值

Structures for propositional logic:

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

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

We call these special interpretations assignments and use *Assign* to denote the set of all assignments.

Example: $AP = \{a, b\}$, $\alpha(a) = 0$, $\alpha(b) = 1$

即对atomatic proposition赋值

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: $AP = \{a, b\}$, $\alpha = \{b\}$ **assignment可以定义为真值集合**

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

Example: $AP = \{a, b\}$, $\alpha = 01$ **如果ap有顺序的话, assignment可以被视为ap的取值序列**

Only the projected assignment matters...

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

Projection

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

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

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

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.

satisfy即assignment取值可使formulae为真

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

A: 16

因为binary operators的输入共有4种类可能: 00, 01, 10, 11

所以对应不同的semantics(即真值表的可能取值)为2^4

同理, three parameter operator 的semantics为2^8

Semantics I: Example

- Let φ be defined as $(a \vee (b \to c))$.
- Let $\alpha : \{a, b, c\} \to \{0, 1\}$ 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	$b \to c$	$a \vee (b \to c)$
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

Satisfaction relation: $\models \subseteq Assign \times PropForm$ Assignment|=PropForm

Instead of $(\alpha, \varphi) \in \models$ we write $\alpha \models \varphi$ and say that 即Assignment使PropForm为真

- α satisfies φ or 若 α |= ф, 则称 α satisfies/is a model of ф, ф holds for α
- φ holds for α or
- α is a $\boxed{\text{model}}$ of φ.

$\models$ is defined recursively:

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

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:

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

Semantics III: The algorithmic view

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

```
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);
}
```

op为运算operation的缩写

- 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

寻找使得PropForm成立的assignment

2^Assign: 所有可能的assignment 情况

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

  PropForm|=2^Assign
  若PropForm成立, 则存在对应的assignment

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

  sat()即为使PropForm取值为真的assignment

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

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

$$\alpha \models \varphi \quad \textit{iff} \quad \alpha \in sat(\varphi)$$

Satisfying assignments: Example

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

Extensions of $\models$

2^Assign: 所有可能的assignment

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

sat()集合中的assignment可以使得φ为真

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

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

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

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

PropForm |= PropForm

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

φ1为真, 则φ2也为真 ==> sat(φ1)属于sat(φ2)

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

Examples: $a \wedge c \models a \vee (b \rightarrow c)$
$x_1 \models x_1 \vee x_2$

Short summary for propositional logic syntax and semantics

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

$$\varphi \ := \ AP \mid (\neg\varphi) \mid (\varphi \wedge \varphi)$$

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

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

  - Satisfaction relation:

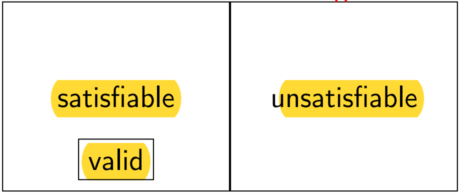
$$\begin{array}{llll} \models \subseteq & \mathit{Assign} \times \mathit{PropForm} & , & (\text{e.g., } \alpha \models \varphi\ ) \\ \models \subseteq & 2^{\mathit{Assign}} \times \mathit{PropForm} & , & (\text{e.g., } \{\alpha_1, \ldots, \alpha_n\} \models \varphi\ ) \\ \models \subseteq & \mathit{PropForm}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

Semantic classification of formulae

- A formula φ is called valid if $sat(\varphi) = Assign$. 永真式valid (任何条件下都为真) 此时sat( )包含所有assignment
  (Also called a tautology).

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

- A formula φ is called unsatisfiable if $sat(\varphi) = \emptyset$.
  (Also called a contradiction). 矛盾式 (任何情况下都为假) 此时sat( )为空



Some notations

- We can write:
  - $\models \varphi$ when φ is valid
  - $\not\models \varphi$ when φ is not valid 不是所有情况都能推出φ
  - $\not\models \neg\varphi$ when φ is satisfiable 不是所有情况都会推出非φ
  - $\models \neg\varphi$ when φ is unsatisfiable

Examples

- $(x_1 \wedge x_2) \rightarrow (x_1 \vee x_2)$ is valid 永真式
- $(x_1 \vee x_2) \rightarrow x_1$ is satisfiable 有成立的情况, 是 satisfiable
- $(x_1 \wedge x_2) \wedge \neg x_1$ is unsatisfiable 没有成立的情况

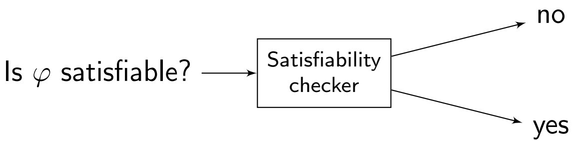
Examples

- Here are some valid formulae:
  - $\models a \land 1 \leftrightarrow a$
  - $\models a \land 0 \leftrightarrow 0$
  - $\models \neg\neg a \leftrightarrow a$ (double-negation rule)
  - $\models a \land (b \lor c) \leftrightarrow (a \land b) \lor (a \land c)$
- Some more (De Morgan rules):
  - $\models \neg(a \land b) \leftrightarrow (\neg a \lor \neg b)$
  - $\models \neg(a \lor b) \leftrightarrow (\neg a \land \neg b)$

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?