Artificial Intelligence

2023/2024 Prof: Sara Bernardini

Lab 7:
DPLL and
First-Order Logic

Francesco Argenziano
email: argenziano@diag.uniroma1.it

*The slides have been prepared using the textbook material available on the web, and the slides of the previous editions of the course by Prof. Luigia Carlucci Aiello, Prof. Daniele Nardi and Dott. Fabio Previtali.

Recall:

- **DPLL** is a complete SAT solver, that given a set of clauses Δ, it either returns that Δ is unsatisfiable, or a partial interpretation I.

- DPLL is an improvement of the Davis-Putnam algorithm, and has laid the foundations of modern SAT solvers for efficient SAT solving.

- Every year there is an annual SAT competition to benchmark the improvements of more and more efficient SAT solvers.

- But have you wondered why SAT is so important to deserve all this attention?

A recap of computational complexity

A decision problem D belongs to (complexity class):

- **P** if it is solvable by a **deterministic** Turing machine in polynomial time (on the size of the input).
  - E.g. Linear Programming

- **NP** if it is solvable in polynomial time by a **nondeterministic** Turing machine.
  - alternatively, if it is *verifiable* in polynomial time by a **deterministic** Turing machine.
  - E.g. SAT

- **co-NP** if its complement $\underline{D}$ is in NP.
  - E.g. TAUT

What?

Let's try to have an intuition of the nature of these problems

- **NP** <—> all those problems of the form:
  - "does it exist (∃) something…"

- **co-NP** <—> all those problems of the form
  - "check if for all (∀) … something is true"

Can you see why SAT is NP and TAUT is co-NP?

Can you see why one is the complement of the other one?

NP

How do we find a solution to a decision problem that is NP (coNP)?

- Remember the truth tables? What we do is a **brute force approach** in listing all the candidate solutions and check if there is at least one (∃) that has the property we are looking for.

- This is the reason why we need a **nondeterministic** Turing machine to solve it in polynomial time.

- **BUT**: we can *verify* if a candidate solution is a solution in polynomial time.
    - E.g. given an interpretation I, it's easy to see if it's a model for a CNF or not.

NP-completeness

A decision problem D is **NP-complete** if:

- it is **NP**
- it is **NP-hard**
  - namely, if **any other problem D'** in NP can be reduced to D in polynomial time.

SAT is **NP-complete**:

- proven in 1970s by the **Cook–Levin theorem**
- before this Thm, the concept of NP-complete did not even exist.

Consequence: **every other NP problem can be rewritten as an instance of a SAT problem**

Let’s think together

What happens if we find out that there is out there an algorithm that is able to **solve** SAT in **polynomial time**?

- Since SAT it’s NP-complete, it automatically means that **all NP** problems are solvable in polynomial time
  - that’s the main core of the **P=NP problem**
- **2-SAT** is solvable in polynomial time
- **SAT** in general we don’t know

Intuition behind trying to find a polynomial time algorithm for an NP problem:

- we want to find a way to make **brute force efficient**

Why do we care?

If a polynomial time algorithm that it's able to solve SAT is found, then arguably our whole society would collapse

Have you ever heard of the **RSA algorithm?**

The RSA algorithm is based on the premise that factoring (a huge semi-prime into two very big prime number) is an NP problem

That's why SAT has all this attention, and why DPLL is so important as a first step towards making SAT solving more efficient.

Exercises: DPLL

For each of the following formulas, **use the DPLL procedure to determine whether it is satisfiable or unsatisfiable**. Transform each formula ϕ_i into an equivalent set of clauses Δ_i. Give a complete trace of the algorithm, **showing the simplified set of clauses for each recursive call of the DPLL function**. Assume that for each rule DPLL selects variables in alphabetical order (i.e., $A, B, C, D, E, \ldots$), and that the **splitting rule first attempts the value False (F) and then the value True (T)** (See Ch 2, slides 14 onwards).

(a) $\phi_1 = (\neg A \vee B \vee C) \wedge (\neg B \vee \neg C) \wedge (\neg A \vee \neg C \vee \neg D) \wedge (C \vee \neg D) \wedge (A \vee D) \wedge (A \vee \neg C \vee \neg D)$

(b) $\phi_2 = (\neg A \vee \neg B \vee C \vee \neg E) \wedge (\neg A \vee \neg B \vee C \vee E) \wedge (A \leftrightarrow B) \wedge (B \vee D) \wedge (B \vee C \vee \neg D) \wedge (\neg C)$

Exercises: DPLL

$\Delta_1 = \{\{\neg A, B, C\}, \{\neg B, \neg C\}, \{\neg A, \neg C, \neg D\}, \{C, \neg D\}, \{A, D\}, \{A, \neg C, \neg D\}\}$

1. Splitting rule:

   1a. $A \mapsto F$
   $\{\{\neg B, \neg C\}, \{C, \neg D\}, \{D\}, \{\neg C, \neg D\}\}$

   2a. Unit propagation: $D \mapsto T$
   $\{\{\neg B, \neg C\}, \{C\}, \{\neg C\}\}$

   3a. Unit propagation: $C \mapsto T$
   $\{\{\neg B\}, \Box\}$

   1b. $A \mapsto T$
   $\{\{B, C\}, \{\neg B, \neg C\}, \{\neg C, \neg D\}, \{C, \neg D\}\}$

   2b. Splitting rule:

      1ba. $B \mapsto F$
      $\{\{C\}, \{\neg C, \neg D\}, \{C, \neg D\}\}$

      2ba. Unit propagation: $C \mapsto T$
      $\{\{\neg D\}\}$

      3ba. Unit propagation: $D \mapsto F$
      $\{\}$

Satisfying assignment: $A, \neg B, C, \neg D$

Exercises: DPLL

Perform DPLL with clause learning.
Start by using the splitting rule and assign the value F to A. For the next splitting rule, assign T to B.
If you encounter a case where two or more different unit propagation rules are applicable choose the one which gets assigned to T.
Whenever you encounter a conflict, mention which clause can be learned with the clause learning method.

$$\Delta = \{\{A, B, C, D\}, \{\neg A, \neg B\}, \{\neg B, \neg C\}, \{\neg A, \neg D\}, \{A, \neg D\}, \{C, \neg D\}, \\ \{B, \neg C\}, \{\neg B, C\}, \{\neg A, C, D\}\}$$

Exercises: DPLL

1. Splitting rule:

   1a. $A \mapsto F$

   $\{\{B, C, D\}, \{\neg B, \neg C\}, \{\neg D\}, \{C, \neg D\}, \{B, \neg C\}, \{\neg B, C\}\}$

   2a. Unit propagation: $D \mapsto F$

   $\{\{B, C\}, \{\neg B, \neg C\}, \{B, \neg C\}, \{\neg B, C\}\}$

   3a. Splitting rule:

   1aa. $B \mapsto T$

   $\{\{\neg C\}, \{C\}\}$

   2aa. Unit propagation: $C \mapsto T$

   $\{\square\}$

   $\rightarrow$ Learned clause: $\neg B$

   i. add $\neg B$ to Δ

   ii. Go back to last splitting rule ($B \mapsto T$)

   iii. Continue: $\{\{B, C\}, \{\neg B, \neg C\}, \{B, \neg C\}, \{\neg B, C\}, \{\neg B\}\}$

Exercises: DPLL

1ab. Unit propagation: $B \mapsto F$

$\{\{C\}, \{\neg C\}\}$

2ab. Unit propagation: $C \mapsto T$

$\{\Box\}$

$\rightarrow$ Learned clause: A

i. add A to Δ

ii. Go back to last splitting rule ($A \mapsto F$)

iii. Continue: $\{\{A, B, C, D\}, \{\neg A, \neg B\}, \{\neg B, \neg C\}, \{\neg A, \neg D\}, \{A, \neg D\}, \{C, \neg D\}, \{B, \neg C\}, \{\neg B, C\}, \{\neg A, C, D\}, \{A\}, \{\neg B\}\}$

1b. Unit propagation: $A \mapsto T$

$\{\{\neg B\}, \{\neg B, \neg C\}, \{\neg D\}, \{C, \neg D\}, \{B, \neg C\}, \{\neg B, C\}, \{C, D\}\}$

2b. Unit propagation: $B \mapsto F$

$\{\{\neg D\}, \{C, \neg D\}, \{\neg C\}, \{C, D\}\}$

2b. Unit propagation: $C \mapsto F$

$\{\{\neg D\}, \{D\}\}$

3b. Unit propagation: $D \mapsto T$

$\{\Box\}$

There is no satisfying assignment.

Intuition with quantifiers

Some worker is a car industry employee

$$\exists x(worker(x) \wedge carIndustryEmployee(x))$$

$$\exists x(worker(x) \Rightarrow carIndustryEmployee(x))$$

All bakers can make appleCakes

$$\forall x(Baker(x) \Rightarrow cando(x, appleCake))$$
$$\forall x(Baker(x) \wedge cando(x, appleCake))$$

Intuition with quantifiers

$\forall x \exists y \; loves(x, y)$ everyone has somebody to love
$\exists x \forall y \; loves(x, y)$ the great lover

Check the use of parameters:

$\forall x \exists y \; loves(y, x)$ somebody loves us
$\exists x \forall y \; loves(y, x)$ the great beloved

FOL properties

1. $\forall x P \equiv \neg \exists x \neg P$
2. $\neg \forall x P \equiv \exists x \neg P$
3. $\exists x P \equiv \neg \forall x \neg P$
4. $\neg \exists x P \equiv \forall x \neg P$.

Quantifiers are distributive wrt $\wedge$ and $\vee$, but with restrictions:

1. $\forall x P_1 \wedge P_2 \equiv (\forall x P_1) \wedge \forall x P_2$ although useless!
2. $\exists x (P_1 \vee P_2) \equiv \exists x P_1 \vee \exists x P_2$ although useless!

3. $\forall x (P_1 \vee P_2) \equiv (\forall x P_1) \vee P_2$ (only) if $x \notin var(P_2)$
4. $\exists x (P_1 \wedge P_2) \equiv (\exists x P_1) \wedge P_2$ (only) if $x \notin var(P_2)$.

FOL properties

Let P_2 a formula where x does not occur free

The **quantifier in the antecedent** changes outside

1. $(\exists x P_1) \Rightarrow P_2 \equiv \forall x(P_1 \Rightarrow P_2)$
2. $(\forall x P_1) \Rightarrow P_2 \equiv \exists x(P_1 \Rightarrow P_2)$

$(\exists x P_1) \Rightarrow P_2$

$\neg(\exists x P_1) \vee P_2$

$(\forall x \neg P_1) \vee P_2$

$\forall x(\neg P_1 \vee P_2)$

$\forall x(P_1 \Rightarrow P_2)$

The **quantifier in the consequent** unchanged outside

3. $P_2 \Rightarrow \exists x P_1 \equiv \exists x(P_2 \Rightarrow P_1)$
4. $P_2 \Rightarrow \forall x P_1 \equiv \forall x(P_2 \Rightarrow P_1)$

Exercises: FOL

1. $\exists x(black(x) \wedge dog(x))$	A. Every dog is black.
2. $\exists x(black(x) \rightarrow dog(x))$	B. There is something black, or a dog.
3. $\exists x(black(x) \vee dog(x))$	C. Everything that is black is a dog.
4. $\forall x(black(x) \wedge dog(x))$	D: Everything is a black dog.
5. $\forall x(black(x) \rightarrow dog(x))$	E. No dog is black.
6. $\forall x(black(x) \vee dog(x))$	F. There is a black dog.

Exercises: FOL

Solution:

1 F

3 B

4 D

5 C

No correspondence for the formulas 2 and 6, and for the sentences A and E.

Exercises: FOL Skolemization

- Is $P(c)$ the Skolemized version of $\exists x\ P(x)$?
- Is $\forall x\ P(c, x)$ the Skolemized version of $\forall x\, \exists y\ P(y, x)$?
- Is $\forall x\ P(f(x), x)$ the Skolemized version of $\forall x\, \exists y\ P(y, x)$?
- Is $P(c_1, c_2)$ the Skolemized version of $\exists x\, \exists y\ P(x, y)$?
- Is $\forall y\ P(c_1, y, f(y))$ the Skolemized version of $\exists x\, \forall y\, \exists z\ P(x, y, z)$?
- Is $P(x, y, f(y))$ the Skolemized version of $\forall x\, \forall y\, \exists z\ P(x, y, z)$?

Exercises: FOL Skolemization

- $\diamond$ Is $P(c)$ the Skolemized version of $\exists x\ P(x)$?
  correct
- $\diamond$ Is $P(c, x)$ the Skolemized version of $\forall x\, \exists y\ P(y, x)$?
  incorrect, see next Skolemization
- $\diamond$ Is $P(f(x), x)$ the Skolemized version of $\forall x\, \exists y\ P(y, x)$?
  correct
- $\diamond$ Is $P(c_1, c_2)$ the Skolemized version of $\exists x\, \exists y\ P(x, y)$?
  correct
- $\diamond$ Is $P(c, y, f(y))$ the Skolemized version of $\exists x\, \forall y\, \exists z\ P(x, y, z)$?
  correct
- $\diamond$ Is $P(x, y, f(y))$ the Skolemized version of $\forall x\, \forall y\, \exists z\ P(x, y, z)$?
  incorrect, z depends on (x, y) hence $f(y) \Rightarrow f(x, y)$

Exercises: FOL Normal Forms

Anyone who kills an animal is loved by no one.

$$\forall x \;\; [\exists y \;\; Animal(y) \land Kills(x, y)] \Rightarrow \forall z \;\; \neg Loves(z, x)$$

There is no rose without a thorn.

$$\forall x (Rose(x) \rightarrow \exists y (Thorn(y) \land Has(x, y)))$$

Exercises: FOL Normal Forms

$$\neg Animal(y) \vee \neg Kills(x, y) \vee \neg Loves(z, x)$$

1a) $\neg Rose(x) \vee Thorn(F(x))$
1b) $\neg Rose(x) \vee Has(x, F(x))$