# Symbolic Artificial Intelligence (COMP3008)
# Decision problems, formal languages and solvers

Daniel Karapetyan
daniel.karapetyan@nottingham.ac.uk

# Long-standing split in AI

| Symbolic AI | Non-symbolic AI |
|---|---|
| Dominant approach until 80-s or 90-s; still active | Dominant since 90-s |
| Knowledge is explicitly represented symbolically | Knowledge is implicit (e.g. training dataset in machine learning) |
| Hand-coded explicit reasoning rules and statements | Knowledge collection is automated |
| Interpretable | Hard to say what a machine-learned program will do next |
| Some tasks which humans can easily do are unlikely to ever be formalisable | Are there things that machine learning will never be able to do? |

# How to solve symbolic AI problems

Need:

- A formal language to describe such problems
- An expert who can formulate a problem in such a formal language
- A software package that can understand the language and solve the problem

# Decision problems

Symbolic AI usually deals with *decision problems*

Decision problem:

- Described symbolically (i.e., via a mathematical model)
- The answer is ‘yes’ or ‘no’
- As opposed to an optimisation problem: the answer is a combination of values of decision variables (solution)

Examples of decision problems:

- Given integer $x$, is $x$ a prime number?
- Given a set of integers, is there a subset of it that adds up to 31?
- Given all the necessary data, can we schedule exams so that there are no conflicts?

# Formal ways to describe a decision problem

*First-order logic*, e.g., is the following expression satisfiable?

$$\forall x.\big(A(x) \rightarrow \exists y.\neg B(x,y)\big)$$

First-order logic is *undecidable*: there is no algorithm that is guaranteed to arrive to an answer to every such question in finite time

An example of a subset of the first-order logic that is decidable:

- *Propositional logic*, e.g., is the following expression satisfiable:

$$(A \wedge \neg B) \rightarrow (\neg A \vee C)$$

Propositional logic is decidable but computationally hard (NP-complete)

# Expressivity vs performance

Trade-off:

**Expressivity:** how rich is the language (how many problems it can describe)

**Performace:** how efficient are the reasoning algorithms

# How bad is undecidability or computational hardness?

Observations:

- Even if a logic is undecidable or computationally hard in general, reasoning may often be manageable in practice
  - Intelligent reasoning algorithm may be able to prioritise the most important choices, etc.
- In some applications, such as automated theorem proving, we may be OK to wait for a very long time, even without guarantee of success, but we need high expressivity (usually full first-order logic)
- In other applications, such as exam timetabling, we may not need the expressivity of first-order logic but we care about the solution time

# SAT Solvers

The SAT problem is a standard form when we use propositional logic:

Given a propositional formula in conjunctive normal form (CNF), find an assignment of propositional variables that satisfies the formula or prove that it is unsatisfiable

(For practical reasons, we want more than just a 'yes'/'no' answer)

- The SAT problem is NP-complete (computationally hard)
- In practice, SAT solvers are extremely efficient
- Some general-purpose solvers actually convert problems into SAT to exploit this efficiency

# DPLL algorithm – background

Most of the modern SAT solvers are based on the DPLL algorithm, named after its creators Davis, Putnam, Logemann and Loveland

- Let $S$ be our expression in CNF
- Observe that $S$ involves a finite number of propositional variables
- We can potentially try every combination of their values
  - If we find a combination that satisfies $S$ then we solved the problem
  - If we did not find any combination that satisfies $S$ then $S$ is unsatisfiable
- In practice, we might not need to enumerate all possible combinations
  - Once we have a partial assignment of values, we may already be able to conclude that this partial assignment does not satisfy $S$
  - For example, in
    $(A \vee \neg B) \wedge (\neg A \vee C) \wedge (D)$
    setting $D =$ FALSE makes the formula unsatisfiable; then there is no need to try all combinations of $A$, $B$ and $C$ with $D =$ FALSE

# DPLL algorithm

- DPLL uses *Depth-First Search*
- It selects a single propositional variable and then splits the search into two branches: when this variable is FALSE and when it is TRUE
- It substitutes the value of the variable and simplifies the formula
- If any clause is FALSE, the formula is unsatisfiable and no further search in this branch is needed
- A few simple heuristics dramatically improve the performance of DPLL

# DPLL algorithm – example

$$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (B \vee C) \wedge (\neg A \vee B \vee C)$$

# DPLL algorithm – example

$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (B \vee C) \wedge (\neg A \vee B \vee C)$
Select $A$:

- Let $A = \text{FALSE}$:
  $(\neg B \vee C) \wedge (B \vee \neg C) \wedge (B \vee C) \wedge (\text{TRUE})$.

# DPLL algorithm – example

$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (B \vee C) \wedge (\neg A \vee B \vee C)$
Select $A$:

- Let $A = \text{FALSE}$:
  $(\neg B \vee C) \wedge (B \vee \neg C) \wedge (B \vee C) \wedge (\text{TRUE})$.
  Select $B$:
  - Let $B = \text{FALSE}$.
    $(\text{TRUE}) \wedge (\neg C) \wedge (C) \wedge (\text{TRUE})$.

# DPLL algorithm – example

$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (B \vee C) \wedge (\neg A \vee B \vee C)$
Select $A$:

- Let $A = \text{FALSE}$:
  $(\neg B \vee C) \wedge (B \vee \neg C) \wedge (B \vee C) \wedge (\text{TRUE})$.
  Select $B$:
  - Let $B = \text{FALSE}$.
    $(\text{TRUE}) \wedge (\neg C) \wedge (C) \wedge (\text{TRUE})$.
    We can set $C = \text{FALSE}$ (heuristic).

# DPLL algorithm – example

$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (B \vee C) \wedge (\neg A \vee B \vee C)$
Select $A$:

- Let $A = \text{FALSE}$:
  $(\neg B \vee C) \wedge (B \vee \neg C) \wedge (B \vee C) \wedge (\text{TRUE})$.
  Select $B$:
  - Let $B = \text{FALSE}$.
    $(\text{TRUE}) \wedge (\neg C) \wedge (C) \wedge (\text{TRUE})$.
    We can set $C = \text{FALSE}$ (heuristic).
    Then $(\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{FALSE}) \wedge (\text{TRUE})$ – this branch is unsatisfiable.

# DPLL algorithm – example

$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (B \vee C) \wedge (\neg A \vee B \vee C)$
Select $A$:

- Let $A = \text{FALSE}$:
  $(\neg B \vee C) \wedge (B \vee \neg C) \wedge (B \vee C) \wedge (\text{TRUE})$.
  Select $B$:
  - Let $B = \text{FALSE}$.
    $(\text{TRUE}) \wedge (\neg C) \wedge (C) \wedge (\text{TRUE})$.
    We can set $C = \text{FALSE}$ (heuristic).
    Then $(\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{FALSE}) \wedge (\text{TRUE})$ – this branch is unsatisfiable.
  - Let $B = \text{TRUE}$.
    $(C) \wedge (\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{TRUE})$.

# DPLL algorithm – example

$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (B \vee C) \wedge (\neg A \vee B \vee C)$
Select $A$:

- Let $A = \text{FALSE}$:
  $(\neg B \vee C) \wedge (B \vee \neg C) \wedge (B \vee C) \wedge (\text{TRUE})$.
  Select $B$:
  - Let $B = \text{FALSE}$.
    $(\text{TRUE}) \wedge (\neg C) \wedge (C) \wedge (\text{TRUE})$.
    We can set $C = \text{FALSE}$ (heuristic).
    Then $(\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{FALSE}) \wedge (\text{TRUE})$ – this branch is unsatisfiable.
  - Let $B = \text{TRUE}$.
    $(C) \wedge (\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{TRUE})$.
    We can set $C = \text{TRUE}$ (heuristic).

# DPLL algorithm – example

$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (B \vee C) \wedge (\neg A \vee B \vee C)$
Select $A$:

- Let $A = \text{FALSE}$:
  $(\neg B \vee C) \wedge (B \vee \neg C) \wedge (B \vee C) \wedge (\text{TRUE})$.
  Select $B$:
  - Let $B = \text{FALSE}$.
    $(\text{TRUE}) \wedge (\neg C) \wedge (C) \wedge (\text{TRUE})$.
    We can set $C = \text{FALSE}$ (heuristic).
    Then $(\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{FALSE}) \wedge (\text{TRUE})$ – this branch is unsatisfiable.
  - Let $B = \text{TRUE}$.
    $(C) \wedge (\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{TRUE})$.
    We can set $C = \text{TRUE}$ (heuristic).
    Then $(\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{TRUE})$ – satisfiable.

# DPLL algorithm – example

$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (B \vee C) \wedge (\neg A \vee B \vee C)$
Select $A$:

- Let $A = \text{FALSE}$:
  $(\neg B \vee C) \wedge (B \vee \neg C) \wedge (B \vee C) \wedge (\text{TRUE})$.
  Select $B$:
  - Let $B = \text{FALSE}$.
    $(\text{TRUE}) \wedge (\neg C) \wedge (C) \wedge (\text{TRUE})$.
    We can set $C = \text{FALSE}$ (heuristic).
    Then $(\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{FALSE}) \wedge (\text{TRUE})$ – this branch is unsatisfiable.
  - Let $B = \text{TRUE}$.
    $(C) \wedge (\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{TRUE})$.
    We can set $C = \text{TRUE}$ (heuristic).
    Then $(\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{TRUE}) \wedge (\text{TRUE})$ – satisfiable.
- We do not need to try $A = \text{TRUE}$ as we have already proven that the problem is satisfiable.

# Summary

- Symbolic AI solves decision problems
- A problem needs to be described in a formal language such as first-order logic
- First-order logic is undecidable in general but some subsets such as propositional logic are decidable
- SAT is a standard form for programs formulated using the propositional logic
- DPLL is the basis of the majority of SAT solvers

# What you will learn in the Symbolic AI module

- How to formulate real-world problems using languages such as SAT
- How to use solvers (Z3 – Microsoft; OR-Tools – Google)
- How those solvers work