Symbolic Artificial Intelligence (COMP3008)
Lecture 6: Propositional Reasoning
3 November 2025

Daniel Karapetyan
daniel.karapetyan@nottingham.ac.uk

Updates

Coursework Project

- Released on 28th October
- There was a minor mistake in the code of the automated test; if you made a copy of the notebook before Friday evening, please update the code of the test
- Deadline 20th November 15:00
- Standard late penalty applies (5 points per day, up to two days)
- Will release the submission system closer to the deadline (to avoid overloading Moodle)
- Expected to spend in the order of 70 hours in total (during the computing classes and in your own time)
- Compulsory interview

Early Module Feedback

- Generally positive
- Pace of delivery seems to be OK
- Many of you found the worksheets helpful and fun
- Many of you found the quizzes helpful
- Requests for more in-class interactive sessions (Slido and review of exercises)
- A few concerns about the worksheets:
  - Do not align with the lecture materials
  - Difficult
  - There were several requests of worksheet solutions
- Immediate changes based on your feedback:
  - Will do more in-class activities
  - I have released the model answers to the worksheets today (see Moodle)

Propositional Logic and CNF

Propositional Logic (Recap)

- Propositional logic is a subset of FOL
- It is 'FOL without the domain of discourse'
- Example of a propositional logic sentence: $P \rightarrow (Q \wedge \neg R)$
- Any propositional formula can be expressed in the standard forms: CNF or DNF
- The above sentence in CNF: $(\neg P \vee Q) \wedge (\neg P \vee \neg R)$

Conversion into CNF

1. Get rid of $\leftrightarrow$ and $\rightarrow$:
   - Replace every $(\alpha \leftrightarrow \beta)$ with $\big((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)\big)$
   - Replace every $(\alpha \rightarrow \beta)$ with $(\neg\alpha \vee \beta)$

   Now, the formula includes only $\wedge$, $\vee$ and $\neg$ operators
2. Push $\neg$ inside brackets:
   - Replace every $\neg(\alpha \wedge \beta \wedge \ldots)$ with $(\neg\alpha \vee \neg\beta \vee \ldots)$
   - Replace every $\neg(\alpha \vee \beta \vee \ldots)$ with $(\neg\alpha \wedge \neg\beta \wedge \ldots)$
   - Replace every $\neg\neg\alpha$ with α

   Now $\neg$ appears only next to literals
3. Example of the formula at this stage:
   $(A \wedge (B \vee (C \wedge \neg A)))) \vee (\neg B \wedge C)$
   - Replace every $((\alpha \wedge \beta) \vee \gamma)$ with $((\alpha \vee \gamma) \wedge (\beta \vee \gamma))$
   - Open unnecessary brackets, e.g. $(\alpha \vee \beta) \vee \gamma$ becomes $\alpha \vee \beta \vee \gamma$
4. Replace every $(\alpha \wedge \alpha)$ and $(\alpha \vee \alpha)$ with α

CNF and DNF size

- Distribution of $\vee$ increases the size of the formula
- E.g. $(X_1 \wedge X_2 \wedge X_3) \vee (Y_1 \wedge Y_2 \wedge Y_3)$
  looks as follows in CNF:
  $(X_1 \vee Y_1) \wedge (X_1 \vee Y_2) \wedge (X_1 \vee Y_3) \wedge (X_2 \vee Y_1) \wedge (X_2 \vee Y_2)$
  $\wedge (X_2 \vee Y_3) \wedge (X_3 \vee Y_1) \wedge (X_3 \vee Y_2) \wedge (X_3 \vee Y_3)$
  Here, the transformation increased the formula size quadratically
- How much can it increase the formula size in the worst case?

CNF and DNF size

- Distribution of $\vee$ increases the size of the formula
- E.g. $(X_1 \wedge X_2 \wedge X_3) \vee (Y_1 \wedge Y_2 \wedge Y_3)$
  looks as follows in CNF:
  $(X_1 \vee Y_1) \wedge (X_1 \vee Y_2) \wedge (X_1 \vee Y_3) \wedge (X_2 \vee Y_1) \wedge (X_2 \vee Y_2)$
  $\wedge (X_2 \vee Y_3) \wedge (X_3 \vee Y_1) \wedge (X_3 \vee Y_2) \wedge (X_3 \vee Y_3)$
  Here, the transformation increased the formula size quadratically
- How much can it increase the formula size in the worst case?

  Some other formulas such as
  $(X_1 \wedge Y_1) \vee (X_2 \wedge Y_2) \vee (X_3 \wedge Y_3) \vee \ldots$
  may increase exponentially when converted into CNF (see `https://en.wikipedia.org/wiki/Conjunctive_normal_form#Conversion_into_CNF` for details)
- This is a problem for reasoning systems that require the formula being in CNF
- DNF has a similar conversion procedure, which may also yield a formula of exponential size

Resolution for Propositional Case

Clausal representation

- Since CNF has a simple structure, it can be written in a compact form called *clausal representation*
- Recall that a formula in CNF is a set of clauses:
  $\underbrace{(A \vee \neg B)}_{\text{Clause 1}} \wedge \underbrace{(\neg C \vee D)}_{\text{Clause 2}} \wedge \underbrace{(\neg A \vee B \vee C)}_{\text{Clause 3}}$
- Each clause is a set of literals, e.g.:
  $(\underbrace{A}_{\text{Literal 1}} \vee \underbrace{\neg B}_{\text{Literal 2}})$
- Each literal is either a predicate or its negation
- We use:
  - Curly brackets for formulas
  - Square brackets for clauses
  - $\overline{\cdot}$ for negation
- E.g., the above formula can be written as
  $\{[A, \overline{B}], [\overline{C}, D], [\overline{A}, B, C]\}$

Clausal representation

- Clausal representation is used for convenience only; it is exactly equivalent to the usual form
- Clausal representation allows us to conveniently handle formulas as mathematical objects (sets of sets)
- [ ] stands for

Clausal representation

- Clausal representation is used for convenience only; it is exactly equivalent to the usual form
- Clausal representation allows us to conveniently handle formulas as mathematical objects (sets of sets)
- [ ] stands for FALSE as the neutral element for disjunction is FALSE: $X \vee \text{FALSE} \leftrightarrow X$
- $\{\}$ stands for

Clausal representation

- Clausal representation is used for convenience only; it is exactly equivalent to the usual form
- Clausal representation allows us to conveniently handle formulas as mathematical objects (sets of sets)
- $[\,]$ stands for FALSE as the neutral element for disjunction is FALSE: $X \vee \text{FALSE} \leftrightarrow X$
- $\{\}$ stands for TRUE as the neutral element for conjunction is TRUE: $X \wedge \text{TRUE} \leftrightarrow X$
- $\{[\,]\}$ stands for

Clausal representation

- Clausal representation is used for convenience only; it is exactly equivalent to the usual form
- Clausal representation allows us to conveniently handle formulas as mathematical objects (sets of sets)
- $[\,]$ stands for FALSE as the neutral element for disjunction is FALSE: $X \vee \text{FALSE} \leftrightarrow X$
- $\{\}$ stands for TRUE as the neutral element for conjunction is TRUE: $X \wedge \text{TRUE} \leftrightarrow X$
- $\{[\,]\}$ stands for FALSE because it is $\{\text{FALSE}\}$

Resolution rule of inference

Let P be a literal and α and β be sets of literals

Having two clauses:
$[P, \alpha]$ and $[\overline{P}, \beta]$
we can *infer* a new clause $[\alpha, \beta]$, called *resolvent*

This process is called *resolution*

Indeed, if P is FALSE then α has to be TRUE; if P is TRUE then β has to be TRUE; hence, either α or β is TRUE

Example

Let us consider two clauses:
$[A, \underbrace{B, \overline{C}}_{\alpha}]$ and $[\overline{A}, \underbrace{D}_{\beta}]$

Then the resolvent is $[B, \overline{C}, D]$, and it is entailed by the two given clauses

Derivation

Note that adding the resolvent $[\alpha, \beta]$ to the formula does not change its evaluation

- Indeed, the resolvent is entailed by the input clauses

Derivation of a clause c from a set of clauses S is a process of inferring clauses $c_1, c_2, \ldots, c_n$, where $c = c_n$ and c_i is inferred from $S \cup \{c_1, c_2, \ldots, c_{i-1}\}$

We write $S \vdash c$ if c can be derived from S

Special case resolvent

For clauses $[P]$ and $[\overline{P}]$, the resolvent is $[\,]$ because $\alpha = \beta = \emptyset$

Recall that $[\,]$ evaluates as FALSE; then clauses $[P]$ and $[\overline{P}]$ entail FALSE

Indeed:
$P \wedge \neg P \models$ FALSE

Derivation and entailment

Recall that the resolvent is entailed by the original clauses

Hence if $S \vdash c$ then $S \models c$

The opposite is not necessarily true, e.g.
$P \models P \vee Q$
but you will never infer $P \vee Q$ from P

In other words, knowing whether $S \vdash \gamma$ is not always enough to know whether $S \models \gamma$

Refutation

However, there is a trick!

It is possible to demonstrate that if S is unsatisfiable then $S \vdash [\,]$

We say that the inference mechanism is *refutation-complete* – it guarantees to prove or disprove that S is unsatisfiable

Negation of the hypothesis

A reasoning system needs to verify if $S \models \gamma$

If we have a refutation-complete inference mechanism, we can use the following observation:

$S \models \gamma$ if and only if $S \wedge \neg\gamma$ is unsatisfiable

Indeed,

- If $S \wedge \neg\gamma$ is unsatisfiable then every interpretation that satisfies S also satisfies γ
- If $S \wedge \neg\gamma$ is satisfiable then there is an interpretation that satisfies S but does not satisfy γ

Then our goal is to verify if $S \cup \{\neg\gamma\}$ is unsatisfiable, which can be done using a refutation-complete inference mechanism

Reasoning system based on resolution

Algorithm 1: Reasoning procedure for propositional logic

input : S (knowledge base in CNF), γ (the hypothesis)

1 Let S' be $\neg\gamma$ in CNF;
2 Let $S \leftarrow S \cup S'$ (i.e. add the negation of the hypothesis to the KB);
3 **Loop**
4 **if** $[\,] \in S$ **then**
5 **return** *ENTAILED*;
6 **if** *exist* $[P, \alpha], [\overline{P}, \beta] \in S$ *such that* $[\alpha, \beta] \notin S$ **then**
7 Add clause $[\alpha, \beta]$ to S;
8 **else**
9 **return** *NOT ENTAILED*;

Summary: reasoning in propositional logic

- FOL is semi-decidable
- To achieve decidability, we can restrict the language: propositional logic
- CNF – to unify the representation
- CNF of a small formula may be exponentially big!
- Resolution rule of inference (one of many possible) but we know it is sufficient
- Every time we apply the resolution rule of inference, we create a new clause that is entailed by our original sentence:
  - Given a sentence (set of sentences, knowledge base) – S
  - By applying the resolution rule of inference, obtain a new clause c
  - Guaranteed that $S \models c$
  - Then $S \wedge c \leftrightarrow S$
  - We can replace S with $S \wedge c$

Refutation-complete reasoning system

- The resolution-based reasoning system may never arrive to our hypothesis
- But it is refutation-complete

  Guaranteed to arrive to an empty clause if the sentence is unsatisfiable
- How can we exploit this to prove or disprove that KB $\models \gamma$?
  - Negate γ
  - Add $\neg\gamma$ to KB
  - Use resolution-based system to check if KB $\wedge \neg\gamma$ infer [ ]
  - If KB $\wedge \neg\gamma \vdash$ [ ] then KB $\wedge \neg\gamma$ is unsatisfiable, hence every interpretation $\mathfrak{I}$ that satisfies S also satisfies γ; we conclude that KB $\models \gamma$
  - If KB $\wedge \neg\gamma \nvdash$ [ ] then KB $\wedge \neg\gamma$ is satisfiable, hence there is an interpretation $\mathfrak{I}$ that satisfies S but does not satisfy γ; we conclude that KB $\not\models \gamma$
  - (This is why we keep revising the definitions of entailment, satisfiability, etc.)

DPLL Algorithm

SAT – recap

- SAT is the problem of finding an interpretation that satisfies a given sentence in CNF (or proving that the sentence is unsatisfiable)
- Can be solved by inference: if $S \vdash []$ then S is unsatisfiable
- If S is satisfiable, can also find a specific interpretation; for example
  - if $S \wedge [A] \vdash []$ then let $A = \text{FALSE}$
  - if $S \wedge [A] \nvdash []$ then let $A = \text{TRUE}$
- However, a purpose-build solver for SAT may (and does in practice) perform better

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 (finite) knowledge base 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 $\{[A, \overline{B}], [\overline{A}, C], [D]\}$ setting $D = \text{FALSE}$ makes the formula unsatisfiable; then there is no need to try all combinations of A, B and C with $D = \text{FALSE}$

DPLL algorithm – concept

- DPLL uses *Depth-First Search* (DFS)
- It selects a single propositional variable and then splits the search into two branches: when this variable is TRUE and when it is FALSE
- Knowing the value of a propositional variable, we can simplify the formula
- If the simplified formula contains an empty clause, it is unsatisfiable and no further search in this branch is needed

DPLL Algorithm

Algorithm 2: DPLL Algorithm (Simplified)

input : Set of clauses S
output: TRUE if S is satisfiable and FALSE otherwise

```
1 if [ ] ∈ S then
2     return FALSE;
3 else if S is empty then
4     return TRUE;
5 else
6     Select a propositional variable A used in S;
7     S' ← Propagate(S, A);
8     S'' ← Propagate(S, Ā);
9     return DPLL(S') ∨ DPLL(S'');
```

The real algorithm also keeps record of the assignments of values to the propositional variables

DPLL Algorithm – Propagation

Algorithm 3: Propagate(S, A)

input : Set of clauses S; literal A (variable or negated variable)
output: Set of clauses S'

```
1  Let S' ← ∅;
2  for c ∈ S do
3      if A ∈ c then
4          Skip this clause (we don't need it as it evaluates to TRUE);
5      else if Ā ∈ c then
6          S' ← S' ∪ {c \ {Ā}};
7          (Since Ā = FALSE, we can remove it but keep the clause)
8      else
9          S' ← S' ∪ {c};
10         (This clause is unaffected by the assignment)
11 return S'
```

Speed Ups of DPLL

DPLL also implements several speed ups

- If a clause consists of one literal only, we can set this literal to true and then propagate
- If a propositional variable appears only in one polarity (i.e. either as A only, or as $\overline{A}$ only), we can set its value (to TRUE or FALSE, respectively) and propagate
- Performance of the algorithm is extremely dependent on the choice of the propositional variable A in line 6 of Algorithm 2; intelligent heuristics are used to select propositional variable A

DPLL Algorithm – example

$$\text{KB:} \qquad \{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$$

Reminder: this is equivalent to
$(A \vee B \vee C) \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg B \vee \neg C)$

DPLL Algorithm – example

$$\text{KB:} \qquad \{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$$

Reminder: this is equivalent to
$(A \vee B \vee C) \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg B \vee \neg C)$

Assume that the branching heuristic selects A:

DPLL Algorithm – example

$$\text{KB:} \qquad \{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$$

Reminder: this is equivalent to
$(A \vee B \vee C) \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg B \vee \neg C)$

Assume that the branching heuristic selects A:
Let $A =$ TRUE:

DPLL Algorithm – example

$$\text{KB:} \quad \{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$$

Reminder: this is equivalent to
$(A \vee B \vee C) \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg B \vee \neg C)$

Assume that the branching heuristic selects A:
Let $A =$ TRUE:

- Propagation: $\{[B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$

DPLL Algorithm – example

$$\text{KB:} \qquad \{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$$

Reminder: this is equivalent to
$(A \vee B \vee C) \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg B \vee \neg C)$

Assume that the branching heuristic selects A:
Let $A =$ TRUE:

- Propagation: $\{[B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
- By speed-up, the first clause consists of only one literal; we set $B =$ TRUE

DPLL Algorithm – example

$$\text{KB:} \qquad \{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$$

Reminder: this is equivalent to
$(A \vee B \vee C) \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg B \vee \neg C)$

Assume that the branching heuristic selects A:
Let $A =$ TRUE:

- Propagation: $\{[B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
- By speed-up, the first clause consists of only one literal; we set $B =$ TRUE
- Propagation: $\{[C],\ [\overline{C}]\}$

DPLL Algorithm – example

$$\text{KB:} \quad \{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$$

Reminder: this is equivalent to
$(A \vee B \vee C) \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg B \vee \neg C)$

Assume that the branching heuristic selects A:
Let $A = \text{TRUE}$:

- Propagation: $\{[B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
- By speed-up, the first clause consists of only one literal; we set $B = \text{TRUE}$
- Propagation: $\{[C],\ [\overline{C}]\}$
- By speed-up, the first clause consists of only one literal; we set $C = \text{TRUE}$

DPLL Algorithm – example

$$\text{KB:} \quad \{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$$

Reminder: this is equivalent to
$(A \vee B \vee C) \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg B \vee \neg C)$

Assume that the branching heuristic selects A:
Let $A = \text{TRUE}$:

- Propagation: $\{[B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
- By speed-up, the first clause consists of only one literal; we set $B = \text{TRUE}$
- Propagation: $\{[C],\ [\overline{C}]\}$
- By speed-up, the first clause consists of only one literal; we set $C = \text{TRUE}$
- Propagation: $\{[\,]\}$

DPLL Algorithm – example

$$\text{KB:} \quad \{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$$

Reminder: this is equivalent to
$(A \vee B \vee C) \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg B \vee \neg C)$

Assume that the branching heuristic selects A:
Let $A = \text{TRUE}$:

- Propagation: $\{[B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
- By speed-up, the first clause consists of only one literal; we set $B = \text{TRUE}$
- Propagation: $\{[C],\ [\overline{C}]\}$
- By speed-up, the first clause consists of only one literal; we set $C = \text{TRUE}$
- Propagation: $\{[\,]\}$
- Conclusion: this branch is unsatisfiable

KB: $\{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$

Let $A =$ FALSE:

KB: $\{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$

Let $A =$ FALSE:

- Propagation: $\{[B, C],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
  Assume that the branching heuristic selects B:
  - Let $B =$ TRUE:
    - Propagation: $\{[C],\ [\overline{C}]\}$

KB: $\{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$

Let $A =$ FALSE:

- Propagation: $\{[B, C],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
  Assume that the branching heuristic selects B:
  - Let $B =$ TRUE:
    - Propagation: $\{[C],\ [\overline{C}]\}$
    - By speed-up, the first clause consists of only one literal; we set $C =$ TRUE

KB: $\{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$

Let $A =$ FALSE:

- Propagation: $\{[B, C],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
  Assume that the branching heuristic selects B:
  - Let $B =$ TRUE:
    - Propagation: $\{[C],\ [\overline{C}]\}$
    - By speed-up, the first clause consists of only one literal; we set $C =$ TRUE
    - Propagation: $\{[\,]\}$

KB: $\{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$

Let $A =$ FALSE:

- Propagation: $\{[B, C],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
  Assume that the branching heuristic selects B:
  - Let $B =$ TRUE:
    - Propagation: $\{[C],\ [\overline{C}]\}$
    - By speed-up, the first clause consists of only one literal; we set $C =$ TRUE
    - Propagation: $\{[\,]\}$
    - Conclusion: this branch is unsatisfiable
  - Let $B =$ FALSE:
    - Propagation: $\{[C]\}$

KB: $\{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$

Let $A =$ FALSE:

- Propagation: $\{[B, C],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
  Assume that the branching heuristic selects B:
  - Let $B =$ TRUE:
    - Propagation: $\{[C],\ [\overline{C}]\}$
    - By speed-up, the first clause consists of only one literal; we set $C =$ TRUE
    - Propagation: $\{[\,]\}$
    - Conclusion: this branch is unsatisfiable
  - Let $B =$ FALSE:
    - Propagation: $\{[C]\}$
    - By speed-up, the first clause consists of only one literal; we set $C =$ TRUE

KB: $\{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$

Let $A =$ FALSE:

- Propagation: $\{[B, C],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
  Assume that the branching heuristic selects B:
  - Let $B =$ TRUE:
    - Propagation: $\{[C],\ [\overline{C}]\}$
    - By speed-up, the first clause consists of only one literal; we set $C =$ TRUE
    - Propagation: $\{[\,]\}$
    - Conclusion: this branch is unsatisfiable
  - Let $B =$ FALSE:
    - Propagation: $\{[C]\}$
    - By speed-up, the first clause consists of only one literal; we set $C =$ TRUE
    - Propagation: $\{\}$

KB: $\{[A, B, C],\ [\overline{A}, B],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$

Let $A =$ FALSE:

- Propagation: $\{[B, C],\ [\overline{B}, C],\ [\overline{B}, \overline{C}]\}$
  Assume that the branching heuristic selects B:
  - Let $B =$ TRUE:
    - Propagation: $\{[C],\ [\overline{C}]\}$
    - By speed-up, the first clause consists of only one literal; we set $C =$ TRUE
    - Propagation: $\{[\,]\}$
    - Conclusion: this branch is unsatisfiable
  - Let $B =$ FALSE:
    - Propagation: $\{[C]\}$
    - By speed-up, the first clause consists of only one literal; we set $C =$ TRUE
    - Propagation: $\{\}$
    - Conclusion: this branch is satisfiable; $A =$ FALSE, $B =$ FALSE, $C =$ TRUE

Reasoning in CSP

CSP solvers

The working of CSP solvers is similar to that of the DPLL algorithm

The power of DPLL comes from propagation

- Once the value of a literal is fixed, we can simplify/remove many clauses

In CSP, propagation is even more powerful:

- Constraints can be more complex and restrictive than SAT clauses
- Once some of the scope variables are assigned values, the solver can simplify the remaining problem

Example of constraint propagation

Consider the following CSP:

- Variables $x_1, x_2 \in \{1, 2, 3\}$ and $x_3 \in \{1, 2\}$
- Constraint Alldiff with scope $\{x_1, x_2, x_3\}$
- Some other variables and constraints...

At some point, the algorithm assigns $x_1 \leftarrow 2$

By propagation of Alldiff, we can immediately conclude that $x_3 = 1$ and $x_2 = 3$

Self-study exercise

Exercise

Convert the following formula into CNF

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

Spoiler alert

Warning: answer in the following slides!

Solution: step 1

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

Solution: step 1

$$(\underbrace{A}_{\alpha} \rightarrow \underbrace{\neg B}_{\beta}) \vee (\neg B \wedge C \wedge D)$$

Replace $(\alpha \rightarrow \beta)$ with $(\neg\alpha \vee \beta)$:

Solution: step 1

$$(\underbrace{A}_{\alpha} \rightarrow \underbrace{\neg B}_{\beta}) \vee (\neg B \wedge C \wedge D)$$

Replace $(\alpha \rightarrow \beta)$ with $(\neg\alpha \vee \beta)$:

$$\alpha = A \qquad \beta = \neg B$$

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

Solution: step 2

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

Solution: step 2

$$\underbrace{(\neg A \vee \neg B)}_{\alpha} \vee (\neg B \wedge C \wedge D)$$

Replace $\alpha \vee (\beta \wedge \gamma)$ with $((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$

Solution: step 2

$$\underbrace{(\neg A \vee \neg B)}_{\alpha} \vee (\neg B \wedge C \wedge D)$$

Replace $\alpha \vee (\beta \wedge \gamma)$ with $((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$

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

Solution: step 3

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

Solution: step 3

$$\big((\neg A \vee \neg B) \vee \neg B\big) \wedge \big(\underbrace{(\neg A \vee \neg B)}_{\alpha} \vee (\underbrace{C}_{\beta} \wedge \underbrace{D}_{\gamma})\big)$$

Replace $\alpha \vee (\beta \wedge \gamma)$ with $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Solution: step 3

$$\big((\neg A \vee \neg B) \vee \neg B\big) \wedge \big(\underbrace{(\neg A \vee \neg B)}_{\alpha} \vee (\underbrace{C}_{\beta} \wedge \underbrace{D}_{\gamma})\big)$$

Replace $\alpha \vee (\beta \wedge \gamma)$ with $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

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

Solution: step 4

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

Solution: step 4

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

Remove unnecessary brackets

Solution: step 4

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

Remove unnecessary brackets

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

Solution: step 5

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

Solution: step 5

$$(\neg A \vee \underbrace{\neg B}_{\alpha} \vee \underbrace{\neg B}_{\alpha}) \wedge (\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee D)$$

Replace $\alpha \vee \alpha$ with α

Solution: step 5

$$(\neg A \vee \underbrace{\neg B}_{\alpha} \vee \underbrace{\neg B}_{\alpha}) \wedge (\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee D)$$

Replace $\alpha \vee \alpha$ with α

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

Solution: step 5

$$(\neg A \vee \underbrace{\neg B}_{\alpha} \vee \underbrace{\neg B}_{\alpha}) \wedge (\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee D)$$

Replace $\alpha \vee \alpha$ with α

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

This formula is in CNF