Chapter 1b

Propositional Logic Review II

(SAT Solving and Application)

Mathematics Modeling

(Materials drawn from **Chapter 1** in:
"Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*, 2nd Ed., Cambridge University Press, 2006"
and some other sources)

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai
Faculty of Computer Science and Engineering
University of Technology, VNU-HCM

Contents

Contents

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Motivated Example – A Logic Puzzle

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

Motivated Example – A Logic Puzzle

- If the unicorn is mythical, then it is immortal;

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

Motivated Example – A Logic Puzzle

- If the unicorn is mythical, then it is immortal;and
- If the unicorn is not mythical, then it is a mortal mammal;

Motivated Example – A Logic Puzzle

- If the unicorn is mythical, then it is immortal;and
- If the unicorn is not mythical, then it is a mortal mammal;and
- If the unicorn is either immortal or a mammal, then it is horned;

Motivated Example – A Logic Puzzle

- If the unicorn is mythical, then it is immortal;and
- If the unicorn is not mythical, then it is a mortal mammal;and
- If the unicorn is either immortal or a mammal, then it is horned;and
- The unicorn is magical if it is horned.

Motivated Example – A Logic Puzzle

- If the unicorn is mythical, then it is immortal;and
- If the unicorn is not mythical, then it is a mortal mammal;and
- If the unicorn is either immortal or a mammal, then it is horned;and
- The unicorn is magical if it is horned.

- **Q:** Is the unicorn mythical? Is it magical? Is it horned?

CNF

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Contents

CNF

- Boolean formula ϕ is defined over a set of propositional variables $p_1, ..., p_n$, using the standard propositional connectives $\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$, and parenthesis

CNF

- Boolean formula ϕ is defined over a set of propositional variables $p_1, ..., p_n$, using the standard propositional connectives $\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{0, 1\}$.

CNF

- Boolean formula ϕ is defined over a set of propositional variables $p_1, ..., p_n$, using the standard propositional connectives $\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{0, 1\}$.
  - Example: $\phi(p_1, p_2, p_3) = ((\neg p_1 \wedge p_2) \vee p_3) \wedge (\neg p_2 \vee p_3)$.

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

CNF

- Boolean formula ϕ is defined over a set of propositional variables $p_1, ..., p_n$, using the standard propositional connectives $\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{0, 1\}$.
  - Example: $\phi(p_1, p_2, p_3) = ((\neg p_1 \wedge p_2) \vee p_3) \wedge (\neg p_2 \vee p_3)$.
- A formula ϕ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement.

CNF

- Boolean formula ϕ is defined over a set of propositional variables $p_1, ..., p_n$, using the standard propositional connectives $\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{0, 1\}$.
  - Example: $\phi(p_1, p_2, p_3) = ((\neg p_1 \wedge p_2) \vee p_3) \wedge (\neg p_2 \vee p_3)$.
- A formula ϕ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement.
  - Example: $\phi(p_1, p_2, p_3) = (\neg p_1 \vee p_2) \wedge (\neg p_2 \vee p_3)$.

CNF

- Boolean formula ϕ is defined over a set of propositional variables $p_1, ..., p_n$, using the standard propositional connectives $\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{0, 1\}$.
  - Example: $\phi(p_1, p_2, p_3) = ((\neg p_1 \wedge p_2) \vee p_3) \wedge (\neg p_2 \vee p_3)$.
- A formula ϕ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement.
  - Example: $\phi(p_1, p_2, p_3) = (\neg p_1 \vee p_2) \wedge (\neg p_2 \vee p_3)$.

Proposition (see [2, Subsection 1.5.1])

There is an algorithm to translate *any* Boolean formula into CNF.

CNF

- Boolean formula ϕ is defined over a set of propositional variables $p_1, ..., p_n$, using the standard propositional connectives $\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{0, 1\}$.
  - Example: $\phi(p_1, p_2, p_3) = ((\neg p_1 \wedge p_2) \vee p_3) \wedge (\neg p_2 \vee p_3)$.
- A formula ϕ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement.
  - Example: $\phi(p_1, p_2, p_3) = (\neg p_1 \vee p_2) \wedge (\neg p_2 \vee p_3)$.

Proposition (see [2, Subsection 1.5.1])

There is an algorithm to translate *any* Boolean formula into CNF.

Proposition 1.45, p. 57

ϕ-satisfiable iff $\neg\phi$-not tautology.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

SAT

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

SAT

Problem

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

SAT

Problem

Find an assignment to the variables $p_1, ..., p_n$ such that $\phi(p_1, ..., p_n) = 1$, or prove that no such assignment exists.

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

SAT

Problem

Find an assignment to the variables $p_1, ..., p_n$ such that $\phi(p_1, ..., p_n) = 1$, or prove that no such assignment exists.

Facts: SAT is an NP-complete decision problem [Cook'71]

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

Contents

SAT

Problem

Find an assignment to the variables $p_1, ..., p_n$ such that $\phi(p_1, ..., p_n) = 1$, or prove that no such assignment exists.

Facts: SAT is an NP-complete decision problem [Cook'71]

- SAT was the first problem to be shown NP-complete.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Contents

SAT

Problem

Find an assignment to the variables $p_1, ..., p_n$ such that $\phi(p_1, ..., p_n) = 1$, or prove that no such assignment exists.

Facts: SAT is an NP-complete decision problem [Cook'71]

- SAT was the first problem to be shown NP-complete.
- There are no known polynomial time algorithms for SAT.

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

SAT

Problem

Find an assignment to the variables $p_1, ..., p_n$ such that $\phi(p_1, ..., p_n) = 1$, or prove that no such assignment exists.

Facts: SAT is an NP-complete decision problem [Cook'71]

- SAT was the first problem to be shown NP-complete.
- There are no known polynomial time algorithms for SAT.
- More-than-35-year old conjecture:

SAT

Problem

Find an assignment to the variables $p_1, ..., p_n$ such that $\phi(p_1, ..., p_n) = 1$, or prove that no such assignment exists.

Facts: SAT is an NP-complete decision problem [Cook'71]

- SAT was the first problem to be shown NP-complete.
- There are no known polynomial time algorithms for SAT.
- More-than-35-year old conjecture:
  "Any algorithm that solves SAT is exponential in the number of variables, in the worst-case."

Polynomial time reductions and NP-Completeness

- Denote

Polynomial time reductions and NP-Completeness

- Denote
  - $EXP = \{\text{Decision problems solvable in exponential time}\}$

Polynomial time reductions and NP-Completeness

- Denote
  - $EXP = \{$Decision problems solvable in exponential time$\}$
  - $P = \{$Decision problems solvable in polynomial time$\}$

Polynomial time reductions and NP-Completeness

- Denote
  - EXP = {Decision problems solvable in exponential time}
  - P = {Decision problems solvable in polynomial time}
  - NP = {Decision problems where Yes solution can verified in polynomial time}

Polynomial time reductions and NP-Completeness

- Denote
  - $EXP = \{$Decision problems solvable in exponential time$\}$
  - $P = \{$Decision problems solvable in polynomial time$\}$
  - $NP = \{$Decision problems where Yes solution can verified in polynomial time$\}$
- A major open question in theoretical computer science is **if** $P = NP$ **or not**.

Polynomial time reductions and NP-Completeness

- Denote
  - $EXP = \{$Decision problems solvable in exponential time$\}$
  - $P = \{$Decision problems solvable in polynomial time$\}$
  - $NP = \{$Decision problems where Yes solution can verified in polynomial time$\}$
- A major open question in theoretical computer science is **if** $P = NP$ **or not**.
- Introduce the notion of **polynomial time reductions** $X \leq_P Y$:

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Polynomial time reductions and NP-Completeness

- Denote
  - EXP = {Decision problems solvable in exponential time}
  - P = {Decision problems solvable in polynomial time}
  - NP = {Decision problems where Yes solution can verified in polynomial time}
- A major open question in theoretical computer science is **if** $P = NP$ **or not**.
- Introduce the notion of **polynomial time reductions** $X \leq_P Y$:

  A problem X is polynomial time reducible to a problem Y ($X \leq_P Y$) if we can solve X in a polynomial number of calls to an algorithm for Y (and the instance of problem Y we solve can be computed in polynomial time from the instance of problem X).

Polynomial time reductions and NP-Completeness

- Denote
  - EXP = {Decision problems solvable in exponential time}
  - P = {Decision problems solvable in polynomial time}
  - NP = {Decision problems where Yes solution can verified in polynomial time}
- A major open question in theoretical computer science is **if** $P = NP$ **or not**.
- Introduce the notion of **polynomial time reductions** $X \leq_P Y$:

  A problem X is polynomial time reducible to a problem Y ($X \leq_P Y$) if we can solve X in a polynomial number of calls to an algorithm for Y (and the instance of problem Y we solve can be computed in polynomial time from the instance of problem X).
- The class of NP-**complete** problems NPC: A problem Y is in NPC if

Polynomial time reductions and NP-Completeness

- Denote
  - $EXP = \{$Decision problems solvable in exponential time$\}$
  - $P = \{$Decision problems solvable in polynomial time$\}$
  - $NP = \{$Decision problems where Yes solution can verified in polynomial time$\}$
- A major open question in theoretical computer science is **if** $P = NP$ **or not**.
- Introduce the notion of **polynomial time reductions** $X \leq_P Y$:

  A problem X is polynomial time reducible to a problem Y ($X \leq_P Y$) if we can solve X in a polynomial number of calls to an algorithm for Y (and the instance of problem Y we solve can be computed in polynomial time from the instance of problem X).
- The class of NP-**complete** problems NPC: A problem Y is in NPC if
  a) $Y \in NP$, and

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

Polynomial time reductions and NP-Completeness

- Denote
  - $EXP = \{$Decision problems solvable in exponential time$\}$
  - $P = \{$Decision problems solvable in polynomial time$\}$
  - $NP = \{$Decision problems where Yes solution can verified in polynomial time$\}$
- A major open question in theoretical computer science is **if** $P = NP$ **or not**.
- Introduce the notion of **polynomial time reductions** $X \leq_P Y$:

  A problem X is polynomial time reducible to a problem Y ($X \leq_P Y$) if we can solve X in a polynomial number of calls to an algorithm for Y (and the instance of problem Y we solve can be computed in polynomial time from the instance of problem X).
- The class of NP-**complete** problems NPC: A problem Y is in NPC if
  a) $Y \in NP$, and
  b) $X \leq_P Y$ for all $X \in NP$.

P=NP question

- The problems in NPC are the hardest problems in NP and the key to resolving the $P = NP$ question.

P=NP question

- The problems in NPC are the hardest problems in NP and the key to resolving the $P = NP$ question.
- If one problem $Y \in NPC$ is in P then $P = NP$.

P=NP question

- The problems in NPC are the hardest problems in NP and the key to resolving the $P = NP$ question.
- If one problem $Y \in NPC$ is in P then $P = NP$.
- If one problem $Y \in NP$ is not in P then $NPC \cap P = \emptyset$.

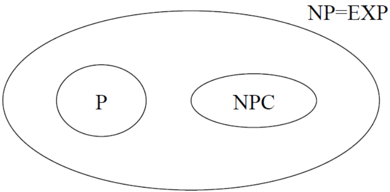
P=NP question

- The problems in NPC are the hardest problems in NP and the key to resolving the $P = NP$ question.
- If one problem $Y \in NPC$ is in P then $P = NP$.
- If one problem $Y \in NP$ is not in P then $NPC \cap P = \emptyset$.
- By now a lot of problems have been proved NP-complete

Contents

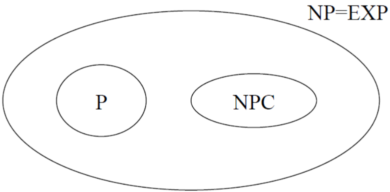
P=NP question

- The problems in NPC are the hardest problems in NP and the key to resolving the $P = NP$ question.
- If one problem $Y \in NPC$ is in P then $P = NP$.
- If one problem $Y \in NP$ is not in P then $NPC \cap P = \emptyset$.
- By now a lot of problems have been proved NP-complete
- We think the world looks like this—but we really do not know:



P=NP question

- The problems in NPC are the hardest problems in NP and the key to resolving the $P = NP$ question.
- If one problem $Y \in NPC$ is in P then $P = NP$.
- If one problem $Y \in NP$ is not in P then $NPC \cap P = \emptyset$.
- By now a lot of problems have been proved NP-complete
- We think the world looks like this—but we really do not know:



- If someone found a polynomial time solution to a problem in NPC our world would "collapse" *and* a lot of smart people have tried really hard to solve NPC problems efficiently

$$\Downarrow$$

We regard $Y \in NPC$ a strong evidence for Y being hard!

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).
- Finding the first problem in NPC is somewhat difficult and require quite a lot of formalism

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).
- Finding the first problem in NPC is somewhat difficult and require quite a lot of formalism
- It seems to be a easier problem 3Sat: Given a formula in 3-CNF, is it satisfiable?

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).
- Finding the first problem in NPC is somewhat difficult and require quite a lot of formalism
- It seems to be a easier problem 3Sat: Given a formula in 3-CNF, is it satisfiable?

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).
- Finding the first problem in NPC is somewhat difficult and require quite a lot of formalism
- It seems to be a easier problem 3Sat: Given a formula in 3-CNF, is it satisfiable?
  - A formula is in 3-CNF (conjunctive normal form) if it consists of an And of 'clauses' each of which is the Or of 3 'literals'

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).
- Finding the first problem in NPC is somewhat difficult and require quite a lot of formalism
- It seems to be a easier problem 3Sat: Given a formula in 3-CNF, is it satisfiable?
  - A formula is in 3-CNF (conjunctive normal form) if it consists of an And of 'clauses' each of which is the Or of 3 'literals'
  - Example: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).
- Finding the first problem in NPC is somewhat difficult and require quite a lot of formalism
- It seems to be a easier problem 3Sat: Given a formula in 3-CNF, is it satisfiable?
  - A formula is in 3-CNF (conjunctive normal form) if it consists of an And of 'clauses' each of which is the Or of 3 'literals'
  - Example: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$
- We prove that 3SAT is in NPC, meaning that it is as hard as general SAT.

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).
- Finding the first problem in NPC is somewhat difficult and require quite a lot of formalism
- It seems to be a easier problem 3Sat: Given a formula in 3-CNF, is it satisfiable?
  - A formula is in 3-CNF (conjunctive normal form) if it consists of an And of 'clauses' each of which is the Or of 3 'literals'
  - Example: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$
- We prove that 3SAT is in NPC, meaning that it is as hard as general SAT.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).
- Finding the first problem in NPC is somewhat difficult and require quite a lot of formalism
- It seems to be a easier problem 3Sat: Given a formula in 3-CNF, is it satisfiable?
  - A formula is in 3-CNF (conjunctive normal form) if it consists of an And of 'clauses' each of which is the Or of 3 'literals'
  - Example: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$
- We prove that 3SAT is in NPC, meaning that it is as hard as general SAT.
  - 3SAT $\in NP$

NP-Complete Problems

- The following lemma helps us to prove a problem NP-complete using another NP-complete problem.

  Lemma: If $Y \in NP$ and $X \leq_P Y$ for some $X \in NPC$ then $Y \in NPC$

  Proof: To prove $Y \in NPC$ we just need to prove $Y \in NP$ (often easy) and reduce problem in NPC to Y (no lower bound proof needed!).
- Finding the first problem in NPC is somewhat difficult and require quite a lot of formalism
- It seems to be a easier problem 3Sat: Given a formula in 3-CNF, is it satisfiable?
  - A formula is in 3-CNF (conjunctive normal form) if it consists of an And of 'clauses' each of which is the Or of 3 'literals'
  - Example: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$
- We prove that 3SAT is in NPC, meaning that it is as hard as general SAT.
  - 3SAT $\in NP$
  - SAT $\leq_P$ 3SAT (we can show that transforming general formula into 3-CNF is in polynomial time.)

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

$$T, \qquad x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad \bar{x}_2.$$

where T denotes the value "Truth".

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

$$T, \qquad x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad \bar{x}_2.$$

  where T denotes the value "Truth".
- We are now *forced* to assign $x_2 = 1$ (as there is a unit-clause), and the formula simplifies to

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

$$T, \qquad x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad \bar{x}_2.$$

  where T denotes the value "Truth".
- We are now *forced* to assign $x_2 = 1$ (as there is a unit-clause), and the formula simplifies to

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

$$T, \qquad x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad \bar{x}_2.$$

  where T denotes the value "Truth".
- We are now *forced* to assign $x_2 = 1$ (as there is a unit-clause), and the formula simplifies to

$$T, \qquad T, \qquad x_3, \qquad x_3 \vee \bar{x}_4, \qquad \emptyset,$$

  where $\emptyset$ is the empty clause which denotes contradiction.

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

$$T, \qquad x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad \bar{x}_2.$$

  where T denotes the value "Truth".
- We are now *forced* to assign $x_2 = 1$ (as there is a unit-clause), and the formula simplifies to

$$T, \qquad T, \qquad x_3, \qquad x_3 \vee \bar{x}_4, \qquad \emptyset,$$

  where $\emptyset$ is the empty clause which denotes contradiction.
- So we have to backtrack to the last *free step*.

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \lor x_2, \qquad x_1 \lor x_2, \qquad \bar{x}_2 \lor x_3, \qquad x_3 \lor \bar{x}_4, \qquad x_1 \lor \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

$$T, \qquad x_2, \qquad \bar{x}_2 \lor x_3, \qquad x_3 \lor \bar{x}_4, \qquad \bar{x}_2.$$

where T denotes the value "Truth".
- We are now *forced* to assign $x_2 = 1$ (as there is a unit-clause), and the formula simplifies to

$$T, \qquad T, \qquad x_3, \qquad x_3 \lor \bar{x}_4, \qquad \emptyset,$$

where $\emptyset$ is the empty clause which denotes contradiction.
- So we have to backtrack to the last *free step*.
- Let's try $x_1 = 1$:

$$x_2, \qquad T, \qquad \bar{x}_2 \lor x_3, \qquad x_3 \lor \bar{x}_4, \qquad T.$$

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

$$T, \qquad x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad \bar{x}_2.$$

  where T denotes the value "Truth".
- We are now *forced* to assign $x_2 = 1$ (as there is a unit-clause), and the formula simplifies to

$$T, \qquad T, \qquad x_3, \qquad x_3 \vee \bar{x}_4, \qquad \emptyset,$$

  where $\emptyset$ is the empty clause which denotes contradiction.
- So we have to backtrack to the last *free step*.
- Let's try $x_1 = 1$:

$$x_2, \qquad T, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad T.$$

- We are now forced to set $x_2 = 1$:

$$T, \qquad T, \qquad x_3, \qquad x_3 \vee \bar{x}_4, \qquad T.$$

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

$$T, \qquad x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad \bar{x}_2.$$

  where T denotes the value "Truth".
- We are now *forced* to assign $x_2 = 1$ (as there is a unit-clause), and the formula simplifies to

$$T, \qquad T, \qquad x_3, \qquad x_3 \vee \bar{x}_4, \qquad \emptyset,$$

  where $\emptyset$ is the empty clause which denotes contradiction.
- So we have to backtrack to the last *free step*.
- Let's try $x_1 = 1$:

$$x_2, \qquad T, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad T.$$

- We are now forced to set $x_2 = 1$:

$$T, \qquad T, \qquad x_3, \qquad x_3 \vee \bar{x}_4, \qquad T.$$

- We are now forced to set $x_3 = 1$:

Example

- Consider the following 2-CNF formula consisting of the following clauses:

$$\bar{x}_1 \vee x_2, \qquad x_1 \vee x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad x_1 \vee \bar{x}_2.$$

- Let's try to set $x_1 = 0$. Then the formula simplifies to:

$$T, \qquad x_2, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad \bar{x}_2.$$

  where T denotes the value "Truth".
- We are now *forced* to assign $x_2 = 1$ (as there is a unit-clause), and the formula simplifies to

$$T, \qquad T, \qquad x_3, \qquad x_3 \vee \bar{x}_4, \qquad \emptyset,$$

  where $\emptyset$ is the empty clause which denotes contradiction.
- So we have to backtrack to the last *free step*.
- Let's try $x_1 = 1$:

$$x_2, \qquad T, \qquad \bar{x}_2 \vee x_3, \qquad x_3 \vee \bar{x}_4, \qquad T.$$

- We are now forced to set $x_2 = 1$:

$$T, \qquad T, \qquad x_3, \qquad x_3 \vee \bar{x}_4, \qquad T.$$

- We are now forced to set $x_3 = 1$:

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

  (a) (Try $x_i = 1$)

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

   (a) (Try $x_i = 1$)

      - Set $\sigma_i = 1$, $\phi' \leftarrow$ Simplify(ϕ, x_i).

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

  (a) (Try $x_i = 1$)

   - Set $\sigma_i = 1$, $\phi' \leftarrow$ Simplify(ϕ, x_i).
   - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

  (a) (Try $x_i = 1$)

   - Set $\sigma_i = 1$, $\phi' \leftarrow$ Simplify(ϕ, x_i).
   - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').
   - If ϕ' does not contain $\emptyset$ goto (1).

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

  (a) (Try $x_i = 1$)

    - Set $\sigma_i = 1$, $\phi' \leftarrow$ Simplify(ϕ, x_i).
    - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').
    - If ϕ' does not contain $\emptyset$ goto (1).

  (b) (Try $x_i = 0$)

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

  (a) (Try $x_i = 1$)
   - Set $\sigma_i = 1$, $\phi' \leftarrow$ Simplify(ϕ, x_i).
   - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').
   - If ϕ' does not contain $\emptyset$ goto (1).

  (b) (Try $x_i = 0$)
   - Unassign variables from step (a).

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

 (a) (Try $x_i = 1$)
 - Set $\sigma_i = 1$, $\phi' \leftarrow$ Simplify(ϕ, x_i).
 - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').
 - If ϕ' does not contain $\emptyset$ goto (1).

 (b) (Try $x_i = 0$)
 - Unassign variables from step (a).
 - Set $\sigma_i = 0$, $\phi' \leftarrow$ Simplify($\phi, \bar{x}_i$).

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.
   - (a) (Try $x_i = 1$)
     - Set $\sigma_i = 1$, $\phi' \leftarrow$ Simplify(ϕ, x_i).
     - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').
     - If ϕ' does not contain $\emptyset$ goto (1).
   - (b) (Try $x_i = 0$)
     - Unassign variables from step (a).
     - Set $\sigma_i = 0$, $\phi' \leftarrow$ Simplify($\phi, \bar{x}_i$).
     - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

<u>Algorithm(ϕ)</u>

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

  (a) (Try $x_i = 1$)
   - Set $\sigma_i = 1$, $\phi' \leftarrow$ Simplify(ϕ, x_i).
   - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').
   - If ϕ' does not contain $\emptyset$ goto (1).

  (b) (Try $x_i = 0$)
   - Unassign variables from step (a).
   - Set $\sigma_i = 0$, $\phi' \leftarrow$ Simplify($\phi, \bar{x}_i$).
   - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').
   - If ϕ' does not contain $\emptyset$ goto (1).

Algorithm(ϕ)

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula ϕ as follows.

Algorithm(ϕ)

(0) Initialize empty assignment $\sigma = *^n$.

(1) If all variables are assigned return σ.

(2) Choose an unassigned variable x_i.

  (a) (Try $x_i = 1$)
  - Set $\sigma_i = 1$, $\phi' \leftarrow$ Simplify(ϕ, x_i).
  - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').
  - If ϕ' does not contain $\emptyset$ goto (1).

  (b) (Try $x_i = 0$)
  - Unassign variables from step (a).
  - Set $\sigma_i = 0$, $\phi' \leftarrow$ Simplify($\phi, \bar{x}_i$).
  - $\phi' \leftarrow$ Unit Clause Propagation(ϕ').
  - If ϕ' does not contain $\emptyset$ goto (1).

(3) Halt with "UNSAT".

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.
  - Otherwise, copy C as is.

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.
  - Otherwise, copy C as is.
- Output the modified formula.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.
  - Otherwise, copy C as is.
- Output the modified formula.

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.
  - Otherwise, copy C as is.
- Output the modified formula.

Unit Clause Propagation(ϕ)

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.
  - Otherwise, copy C as is.
- Output the modified formula.

Unit Clause Propagation(ϕ)

- While $\exists$ unit clause ℓ_i:

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Simplify(ϕ, ℓ_i)

<u>Simplify(ϕ, ℓ_i)</u>

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.
  - Otherwise, copy C as is.
- Output the modified formula.

<u>Unit Clause Propagation(ϕ)</u>

- While $\exists$ unit clause ℓ_i:
  - Update σ: if $\ell_i = x_i$ set $\sigma_i = 1$, else ($\ell_i = \bar{x}_i$) set $\sigma_i = 0$.

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.
  - Otherwise, copy C as is.
- Output the modified formula.

Unit Clause Propagation(ϕ)

- While $\exists$ unit clause ℓ_i:
  - Update σ: if $\ell_i = x_i$ set $\sigma_i = 1$, else ($\ell_i = \bar{x}_i$) set $\sigma_i = 0$.
  - $\phi \leftarrow$Simplify(ϕ, ℓ_i).

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.
  - Otherwise, copy C as is.
- Output the modified formula.

Unit Clause Propagation(ϕ)

- While $\exists$ unit clause ℓ_i:
  - Update σ: if $\ell_i = x_i$ set $\sigma_i = 1$, else ($\ell_i = \bar{x}_i$) set $\sigma_i = 0$.
  - $\phi \leftarrow$Simplify(ϕ, ℓ_i).

Simplify(ϕ, ℓ_i)

Simplify(ϕ, ℓ_i)

- $\forall$ clause $C \in \phi$:
  - If $\ell_i \in C$, remove C.
  - If $\bar{\ell}_i \in C$, $C \leftarrow C \setminus \bar{\ell}_i$.
  - Otherwise, copy C as is.
- Output the modified formula.

Unit Clause Propagation(ϕ)

- While $\exists$ unit clause ℓ_i:
  - Update σ: if $\ell_i = x_i$ set $\sigma_i = 1$, else ($\ell_i = \bar{x}_i$) set $\sigma_i = 0$.
  - $\phi \leftarrow$Simplify(ϕ, ℓ_i).

Complexity: Let n denote the number of variables and let m denote the number of clauses. It is not hard to verify that there are at most n outer iterations and that each call to UCP takes at most $O(m)$ time, therefore the running time of Algorithm is $O(m \cdot n)$. (HW: Find an implementation in $O(n + m)$ complexity.)

Correctness of the Algorithm

Lemma

If the algorithm outputs an assignment σ, then σ satisfies ϕ.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Contents

Correctness of the Algorithm

Lemma

If the algorithm outputs an assignment σ, then σ satisfies ϕ.

We will need the following definition: A partial assignment $\sigma \in \{0, 1, *\}^n$ *violate* a clause $C = \ell_i \vee \ell_j$ if: σ_i and σ_j are assigned (i.e.,$\sigma_i, \sigma_j \neq *$) and σ_i doesn't satisfy ℓ_i and σ_j doesn't satisfy ℓ_j.

Correctness of the Algorithm

Lemma

If the algorithm outputs an assignment σ, then σ satisfies ϕ.

We will need the following definition: A partial assignment $\sigma \in \{0, 1, *\}^n$ *violate* a clause $C = \ell_i \vee \ell_j$ if: σ_i and σ_j are assigned (i.e.,$\sigma_i, \sigma_j \neq *$) and σ_i doesn't satisfy ℓ_i and σ_j doesn't satisfy ℓ_j. The lemma follows from the following invariance.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Correctness of the Algorithm

Lemma

If the algorithm outputs an assignment σ, then σ satisfies ϕ.

We will need the following definition: A partial assignment $\sigma \in \{0, 1, *\}^n$ *violate* a clause $C = \ell_i \vee \ell_j$ if: σ_i and σ_j are assigned (i.e.,$\sigma_i, \sigma_j \neq *$) and σ_i doesn't satisfy ℓ_i and σ_j doesn't satisfy ℓ_j. The lemma follows from the following invariance.

Lemma

At the beginning of each iteration, the current partial assignment $\sigma^{(i)}$ does not violate any of the clauses of C.

Correctness of the Algorithm

Lemma

If the algorithm outputs an assignment σ, then σ satisfies ϕ.

We will need the following definition: A partial assignment $\sigma \in \{0, 1, *\}^n$ *violate* a clause $C = \ell_i \vee \ell_j$ if: σ_i and σ_j are assigned (i.e.,$\sigma_i, \sigma_j \neq *$) and σ_i doesn't satisfy ℓ_i and σ_j doesn't satisfy ℓ_j. The lemma follows from the following invariance.

Lemma

At the beginning of each iteration, the current partial assignment $\sigma^{(i)}$ does not violate any of the clauses of C.

Chứng minh.

Invariance 2 By induction on i. The basis is trivial as in the first iteration $\sigma = *^n$ and so none of the clauses are violated.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Correctness of the Algorithm

Lemma

If the algorithm outputs an assignment σ, then σ satisfies ϕ.

We will need the following definition: A partial assignment $\sigma \in \{0, 1, *\}^n$ *violate* a clause $C = \ell_i \vee \ell_j$ if: σ_i and σ_j are assigned (i.e.,$\sigma_i, \sigma_j \neq *$) and σ_i doesn't satisfy ℓ_i and σ_j doesn't satisfy ℓ_j. The lemma follows from the following invariance.

Lemma

At the beginning of each iteration, the current partial assignment $\sigma^{(i)}$ does not violate any of the clauses of C.

Chứng minh.

Invariance 2 By induction on i. The basis is trivial as in the first iteration $\sigma = *^n$ and so none of the clauses are violated. Step: we'll prove that none of the clauses C are violated by $\sigma^{(i+1)}$. If both variables of C were assigned before the last iteration, then, by the induction hypothesis, $\sigma^{(i)}$ doesn't violate C, and therefore, so is $\sigma^{(i+1)}$. If both variables of C were assigned in the last iteration, then C must be satisfied by $\sigma^{(i+1)}$, otherwise, the algorithm finds a contradiction. □

Correctness of the Algorithm (cont.)

Lemma

If the algorithm outputs UNSAT, then ϕ is unsatisfiable.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Correctness of the Algorithm (cont.)

Lemma

If the algorithm outputs UNSAT, then ϕ is unsatisfiable.

Chứng minh.

- Let ϕ' be the formula at the beginning of the iteration in which A halts, and let x_i be the variable chosen at step (2) of this last iteration.

$\square$

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Correctness of the Algorithm (cont.)

Lemma

If the algorithm outputs UNSAT, then ϕ is unsatisfiable.

Chứng minh.

- Let ϕ' be the formula at the beginning of the iteration in which A halts, and let x_i be the variable chosen at step (2) of this last iteration.
- Note that ϕ' is a 2-CNF formula and $\phi' \subseteq \phi$ (i.e., all the clauses of ϕ' appear as clauses in ϕ).

□

Correctness of the Algorithm (cont.)

Lemma

If the algorithm outputs UNSAT, then ϕ is unsatisfiable.

Chứng minh.

- Let ϕ' be the formula at the beginning of the iteration in which A halts, and let x_i be the variable chosen at step (2) of this last iteration.
- Note that ϕ' is a 2-CNF formula and $\phi' \subseteq \phi$ (i.e., all the clauses of ϕ' appear as clauses in ϕ).
- Hence, it suffices to show that ϕ' is unsatisfiable.

□

Correctness of the Algorithm (cont.)

Lemma

If the algorithm outputs UNSAT, then ϕ is unsatisfiable.

Chứng minh.

- Let ϕ' be the formula at the beginning of the iteration in which A halts, and let x_i be the variable chosen at step (2) of this last iteration.
- Note that ϕ' is a 2-CNF formula and $\phi' \subseteq \phi$ (i.e., all the clauses of ϕ' appear as clauses in ϕ).
- Hence, it suffices to show that ϕ' is unsatisfiable.
- Let ϕ_0=Simplify($\phi', x_i = 0$) and ϕ_1=Simplify($\phi', x_i = 1$). It suffices to show that both ϕ_0 and ϕ_1 are unsatisfiable.

□

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Correctness of the Algorithm (cont.)

Lemma

If the algorithm outputs UNSAT, then ϕ is unsatisfiable.

Chứng minh.

- Let ϕ' be the formula at the beginning of the iteration in which A halts, and let x_i be the variable chosen at step (2) of this last iteration.
- Note that ϕ' is a 2-CNF formula and $\phi' \subseteq \phi$ (i.e., all the clauses of ϕ' appear as clauses in ϕ).
- Hence, it suffices to show that ϕ' is unsatisfiable.
- Let ϕ_0=Simplify($\phi', x_i = 0$) and ϕ_1=Simplify($\phi', x_i = 1$). It suffices to show that both ϕ_0 and ϕ_1 are unsatisfiable.
- Recall that the formula UCP(ϕ_0) and the formula UCP(ϕ_1) contain a contradiction.

□

Correctness of the Algorithm (cont.)

Lemma

If the algorithm outputs UNSAT, then ϕ is unsatisfiable.

Chứng minh.

- Let ϕ' be the formula at the beginning of the iteration in which A halts, and let x_i be the variable chosen at step (2) of this last iteration.
- Note that ϕ' is a 2-CNF formula and $\phi' \subseteq \phi$ (i.e., all the clauses of ϕ' appear as clauses in ϕ).
- Hence, it suffices to show that ϕ' is unsatisfiable.
- Let ϕ_0=Simplify($\phi', x_i = 0$) and ϕ_1=Simplify($\phi', x_i = 1$). It suffices to show that both ϕ_0 and ϕ_1 are unsatisfiable.
- Recall that the formula UCP(ϕ_0) and the formula UCP(ϕ_1) contain a contradiction.
- The proof now follows by noting that if UCP(ψ) contains a contradiction, then ψ is UNSAT.

□

Correctness of the Algorithm (cont.)

Lemma

If the algorithm outputs UNSAT, then ϕ is unsatisfiable.

Chứng minh.

- Let ϕ' be the formula at the beginning of the iteration in which A halts, and let x_i be the variable chosen at step (2) of this last iteration.
- Note that ϕ' is a 2-CNF formula and $\phi' \subseteq \phi$ (i.e., all the clauses of ϕ' appear as clauses in ϕ).
- Hence, it suffices to show that ϕ' is unsatisfiable.
- Let ϕ_0=Simplify($\phi', x_i = 0$) and ϕ_1=Simplify($\phi', x_i = 1$). It suffices to show that both ϕ_0 and ϕ_1 are unsatisfiable.
- Recall that the formula UCP(ϕ_0) and the formula UCP(ϕ_1) contain a contradiction.
- The proof now follows by noting that if UCP(ψ) contains a contradiction, then ψ is UNSAT.

□

Correctness of the Algorithm (cont.)

Lemma

If the algorithm outputs UNSAT, then ϕ is unsatisfiable.

Chứng minh.

- Let ϕ' be the formula at the beginning of the iteration in which A halts, and let x_i be the variable chosen at step (2) of this last iteration.
- Note that ϕ' is a 2-CNF formula and $\phi' \subseteq \phi$ (i.e., all the clauses of ϕ' appear as clauses in ϕ).
- Hence, it suffices to show that ϕ' is unsatisfiable.
- Let ϕ_0=Simplify($\phi', x_i = 0$) and ϕ_1=Simplify($\phi', x_i = 1$). It suffices to show that both ϕ_0 and ϕ_1 are unsatisfiable.
- Recall that the formula UCP(ϕ_0) and the formula UCP(ϕ_1) contain a contradiction.
- The proof now follows by noting that if UCP(ψ) contains a contradiction, then ψ is UNSAT.

□

Therefore, we have an efficient algorithm for SAT of 2-CNF.

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:
  - nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:
  - nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$
  - for a clause $\ell_i \vee \ell_j$ define the edges:

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:
  - nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$
  - for a clause $\ell_i \vee \ell_j$ define the edges:

    $\bar{\ell}_i \to \ell_j$

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:
  - nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$
  - for a clause $\ell_i \vee \ell_j$ define the edges:

    $\bar{\ell}_i \rightarrow \ell_j$
    $\bar{\ell}_j \rightarrow \ell_i$

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:
  - nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$
  - for a clause $\ell_i \vee \ell_j$ define the edges:

    $\bar{\ell}_i \rightarrow \ell_j$
    $\bar{\ell}_j \rightarrow \ell_i$

  Main property: Let σ be a satisfying assignment.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:
  - nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$
  - for a clause $\ell_i \vee \ell_j$ define the edges:

    $\bar{\ell}_i \to \ell_j$
    $\bar{\ell}_j \to \ell_i$

  Main property: Let σ be a satisfying assignment.

  If σ satisfies a node v, then σ satisfies all nodes u achievable from v.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:
  - nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$
  - for a clause $\ell_i \vee \ell_j$ define the edges:

    $\bar{\ell}_i \to \ell_j$
    $\bar{\ell}_j \to \ell_i$

Main property: Let σ be a satisfying assignment.

If σ satisfies a node v, then σ satisfies all nodes u achievable from v.

The property can be proven by induction on the length of the path.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Contents

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:
  - nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$
  - for a clause $\ell_i \vee \ell_j$ define the edges:

    $\bar{\ell}_i \rightarrow \ell_j$
    $\bar{\ell}_j \rightarrow \ell_i$

  Main property: Let σ be a satisfying assignment.

  If σ satisfies a node v, then σ satisfies all nodes u achievable from v.

  The property can be proven by induction on the length of the path.

Contents

Graphical View of 2-SAT

- For a 2-CNF formula ϕ, define the implication graph $G = G_\phi$ as follows:
  - nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$
  - for a clause $\ell_i \vee \ell_j$ define the edges:

    $\bar{\ell}_i \to \ell_j$
    $\bar{\ell}_j \to \ell_i$

<u>Main property</u>: Let σ be a satisfying assignment.

If σ satisfies a node v, then σ satisfies all nodes u achievable from v.

The property can be proven by induction on the length of the path.

Theorem

ϕ is satisfiable iff the graph G does not contain a "contradiction path" of the form:

$$\ell_i \to \cdots \to \bar{\ell}_i \to \cdots \to \ell_i.$$

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.
2. (ϕ is UNSAT $\Rightarrow \exists$ contradiction path):
   If ϕ is UNSAT $\Rightarrow$ Algorithm Halts.
   $\Rightarrow$ for some x_i we have:

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.
2. (ϕ is UNSAT $\Rightarrow \exists$ contradiction path):
   If ϕ is UNSAT $\Rightarrow$ Algorithm Halts.
   $\Rightarrow$ for some x_i we have:
   (a) $\ell_j \leftarrow \cdots \leftarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j$

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.
2. (ϕ is UNSAT $\Rightarrow \exists$ contradiction path):
   If ϕ is UNSAT $\Rightarrow$ Algorithm Halts.
   $\Rightarrow$ for some x_i we have:
   (a) $\ell_j \leftarrow \cdots \leftarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j$
   (b) $\ell_k \leftarrow \cdots \leftarrow \bar{x}_i \rightarrow \cdots \rightarrow \bar{\ell}_k$

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.
2. (ϕ is UNSAT $\Rightarrow \exists$ contradiction path):
   If ϕ is UNSAT $\Rightarrow$ Algorithm Halts.
   $\Rightarrow$ for some x_i we have:
   (a) $\ell_j \leftarrow \cdots \leftarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j$
   (b) $\ell_k \leftarrow \cdots \leftarrow \bar{x}_i \rightarrow \cdots \rightarrow \bar{\ell}_k$

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.
2. (ϕ is UNSAT $\Rightarrow \exists$ contradiction path):
   If ϕ is UNSAT $\Rightarrow$ Algorithm Halts.
   $\Rightarrow$ for some x_i we have:
   (a) $\ell_j \leftarrow \cdots \leftarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j$
   (b) $\ell_k \leftarrow \cdots \leftarrow \bar{x}_i \rightarrow \cdots \rightarrow \bar{\ell}_k$

   In our graph, if $\ell_i \rightarrow \ell_j$ is an edge, then $\bar{\ell}_j \rightarrow \bar{\ell}_i$ is also an edge.

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.
2. (ϕ is UNSAT $\Rightarrow \exists$ contradiction path):
   If ϕ is UNSAT $\Rightarrow$ Algorithm Halts.
   $\Rightarrow$ for some x_i we have:
   (a) $\ell_j \leftarrow \cdots \leftarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j$
   (b) $\ell_k \leftarrow \cdots \leftarrow \bar{x}_i \rightarrow \cdots \rightarrow \bar{\ell}_k$

   In our graph, if $\ell_i \rightarrow \ell_j$ is an edge, then $\bar{\ell}_j \rightarrow \bar{\ell}_i$ is also an edge.

   By reversing edges and negating:

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.
2. (ϕ is UNSAT $\Rightarrow \exists$ contradiction path):
   If ϕ is UNSAT $\Rightarrow$ Algorithm Halts.
   $\Rightarrow$ for some x_i we have:
   (a) $\ell_j \leftarrow \cdots \leftarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j$
   (b) $\ell_k \leftarrow \cdots \leftarrow \bar{x}_i \rightarrow \cdots \rightarrow \bar{\ell}_k$

   In our graph, if $\ell_i \rightarrow \ell_j$ is an edge, then $\bar{\ell}_j \rightarrow \bar{\ell}_i$ is also an edge.

   By reversing edges and negating:
   (a) $\Rightarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j \rightarrow \cdots \rightarrow \bar{x}_i$

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.
2. (ϕ is UNSAT $\Rightarrow \exists$ contradiction path):
   If ϕ is UNSAT $\Rightarrow$ Algorithm Halts.
   $\Rightarrow$ for some x_i we have:
   (a) $\ell_j \leftarrow \cdots \leftarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j$
   (b) $\ell_k \leftarrow \cdots \leftarrow \bar{x}_i \rightarrow \cdots \rightarrow \bar{\ell}_k$

   In our graph, if $\ell_i \rightarrow \ell_j$ is an edge, then $\bar{\ell}_j \rightarrow \bar{\ell}_i$ is also an edge.

   By reversing edges and negating:
   (a) $\Rightarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j \rightarrow \cdots \rightarrow \bar{x}_i$
   (b) $\Rightarrow \bar{x}_i \rightarrow \cdots \rightarrow \bar{\ell}_k \rightarrow \cdots \rightarrow x_i$

Proof for the previous theorem

1. ($\exists$ contradiction path $\Rightarrow \phi$ is UNSAT):
   - Take a potential assignment σ.
   - If σ satisfies ℓ_i, then by Property it must satisfy $\bar{\ell}_i$. Contradiction.
   - If σ satisfies $\bar{\ell}_i$, then by Property it must satisfy ℓ_i. Contradiction.
2. (ϕ is UNSAT $\Rightarrow \exists$ contradiction path):
   If ϕ is UNSAT $\Rightarrow$ Algorithm Halts.
   $\Rightarrow$ for some x_i we have:
   (a) $\ell_j \leftarrow \cdots \leftarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j$
   (b) $\ell_k \leftarrow \cdots \leftarrow \bar{x}_i \rightarrow \cdots \rightarrow \bar{\ell}_k$

   In our graph, if $\ell_i \rightarrow \ell_j$ is an edge, then $\bar{\ell}_j \rightarrow \bar{\ell}_i$ is also an edge.

   By reversing edges and negating:
   (a) $\Rightarrow x_i \rightarrow \cdots \rightarrow \bar{\ell}_j \rightarrow \cdots \rightarrow \bar{x}_i$
   (b) $\Rightarrow \bar{x}_i \rightarrow \cdots \rightarrow \bar{\ell}_k \rightarrow \cdots \rightarrow x_i$

   Therefore, there exists a contradiction path.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

1 Introduction

2 2-SAT is in P

3 SAT Solvers

- WalkSAT: Idea
- DPLL: Idea
- A Linear Solver
- A Cubic Solver

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.
- **Details:** In each step, choose an unsatisfied clause (clause selection), and "flip" one of its variables (variable selection).

Contents

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.
- **Details:** In each step, choose an unsatisfied clause (clause selection), and "flip" one of its variables (variable selection).

WalkSAT: Details

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.
- **Details:** In each step, choose an unsatisfied clause (clause selection), and "flip" one of its variables (variable selection).

WalkSAT: Details

- **Termination criterion:** No unsatisfied clauses are left.

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.
- **Details:** In each step, choose an unsatisfied clause (clause selection), and "flip" one of its variables (variable selection).

WalkSAT: Details

- **Termination criterion:** No unsatisfied clauses are left.
- **Clause selection:** Choose a random unsatisfied clause.

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.
- **Details:** In each step, choose an unsatisfied clause (clause selection), and "flip" one of its variables (variable selection).

WalkSAT: Details

- **Termination criterion:** No unsatisfied clauses are left.
- **Clause selection:** Choose a random unsatisfied clause.
- **Variable selection:**

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.
- **Details:** In each step, choose an unsatisfied clause (clause selection), and "flip" one of its variables (variable selection).

WalkSAT: Details

- **Termination criterion:** No unsatisfied clauses are left.
- **Clause selection:** Choose a random unsatisfied clause.
- **Variable selection:**
  - If there are variables that when flipped make no currently satisfied clause unsatisfied, flip one which makes the most unsatisfied clauses satisfied.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.
- **Details:** In each step, choose an unsatisfied clause (clause selection), and "flip" one of its variables (variable selection).

WalkSAT: Details

- **Termination criterion:** No unsatisfied clauses are left.
- **Clause selection:** Choose a random unsatisfied clause.
- **Variable selection:**
  - If there are variables that when flipped make no currently satisfied clause unsatisfied, flip one which makes the most unsatisfied clauses satisfied.
  - Otherwise, make a choice with a certain probability between:

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.
- **Details:** In each step, choose an unsatisfied clause (clause selection), and "flip" one of its variables (variable selection).

WalkSAT: Details

- **Termination criterion:** No unsatisfied clauses are left.
- **Clause selection:** Choose a random unsatisfied clause.
- **Variable selection:**
  - If there are variables that when flipped make no currently satisfied clause unsatisfied, flip one which makes the most unsatisfied clauses satisfied.
  - Otherwise, make a choice with a certain probability between:
    - picking a random variable, and

WalkSAT: An Incomplete Solver

- **Idea:** Start with a random truth assignment, and then iteratively improve the assignment until model is found.
- **Details:** In each step, choose an unsatisfied clause (clause selection), and "flip" one of its variables (variable selection).

WalkSAT: Details

- **Termination criterion:** No unsatisfied clauses are left.
- **Clause selection:** Choose a random unsatisfied clause.
- **Variable selection:**
  - If there are variables that when flipped make no currently satisfied clause unsatisfied, flip one which makes the most unsatisfied clauses satisfied.
  - Otherwise, make a choice with a certain probability between:
    - picking a random variable, and
    - picking a variable that when flipped minimizes the number of unsatisfied clauses.

DPLL: Idea

- Simplify formula based on

DPLL: Idea

- Simplify formula based on pure literal elimination and

DPLL: Idea

- Simplify formula based on pure literal elimination and unit propagation

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

DPLL: Idea

- Simplify formula based on pure literal elimination and unit propagation
- If not done, pick an atom p and split: $\phi \wedge p$ or $\phi \wedge \neg p$

A Linear Solver: Idea

- Transform formula to tree of conjunctions and negations.

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

A Linear Solver: Idea

- Transform formula to tree of conjunctions and negations.
- Transform tree into graph.

A Linear Solver: Idea

- Transform formula to tree of conjunctions and negations.
- Transform tree into graph.
- Mark the top of the tree as T.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

A Linear Solver: Idea

- Transform formula to tree of conjunctions and negations.
- Transform tree into graph.
- Mark the top of the tree as T.
- Propagate constraints using obvious rules.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

A Linear Solver: Idea

- Transform formula to tree of conjunctions and negations.
- Transform tree into graph.
- Mark the top of the tree as `T`.
- Propagate constraints using obvious rules.
- If all leaves are marked, check that corresponding assignment makes the formula true.

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

Transformation

$$T(p) = p$$

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

Transformation

$$\begin{aligned} T(p) &= p \\ T(\phi_1 \wedge \phi_2) &= T(\phi_1) \wedge T(\phi_2) \end{aligned}$$

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Transformation

$$
\begin{aligned}
T(p) &= p \\
T(\phi_1 \wedge \phi_2) &= T(\phi_1) \wedge T(\phi_2) \\
T(\neg\phi) &= \neg\phi(T)
\end{aligned}
$$

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Transformation

$$
\begin{array}{rcl}
T(p) & = & p \\
T(\phi_1 \wedge \phi_2) & = & T(\phi_1) \wedge T(\phi_2) \\
T(\neg\phi) & = & \neg\phi(T) \\
T(\phi_1 \rightarrow \phi_2) & = & \neg(T(\phi_1) \wedge \neg T(\phi_2))
\end{array}
$$

Transformation

$$
\begin{aligned}
T(p) &= p \\
T(\phi_1 \wedge \phi_2) &= T(\phi_1) \wedge T(\phi_2) \\
T(\neg\phi) &= \neg\phi(T) \\
T(\phi_1 \rightarrow \phi_2) &= \neg(T(\phi_1) \wedge \neg T(\phi_2)) \\
T(\phi_1 \vee \phi_2) &= \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))
\end{aligned}
$$

Transformation

$$\begin{aligned}
T(p) &= p \\
T(\phi_1 \wedge \phi_2) &= T(\phi_1) \wedge T(\phi_2) \\
T(\neg\phi) &= \neg\phi(T) \\
T(\phi_1 \rightarrow \phi_2) &= \neg(T(\phi_1) \wedge \neg T(\phi_2)) \\
T(\phi_1 \vee \phi_2) &= \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))
\end{aligned}$$

Example

Transformation

$$
\begin{aligned}
T(p) &= p \\
T(\phi_1 \wedge \phi_2) &= T(\phi_1) \wedge T(\phi_2) \\
T(\neg\phi) &= \neg\phi(T) \\
T(\phi_1 \rightarrow \phi_2) &= \neg(T(\phi_1) \wedge \neg T(\phi_2)) \\
T(\phi_1 \vee \phi_2) &= \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))
\end{aligned}
$$

Example

$$\phi = p \wedge \neg(q \vee \neg p)$$

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Contents

Transformation

$$
\begin{aligned}
T(p) &= p \\
T(\phi_1 \wedge \phi_2) &= T(\phi_1) \wedge T(\phi_2) \\
T(\neg\phi) &= \neg\phi(T) \\
T(\phi_1 \rightarrow \phi_2) &= \neg(T(\phi_1) \wedge \neg T(\phi_2)) \\
T(\phi_1 \vee \phi_2) &= \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))
\end{aligned}
$$

Example

$$\phi = p \wedge \neg(q \vee \neg p)$$

$$T(\phi) = p \wedge \neg\neg(\neg q \wedge \neg\neg p)$$

Binary Decision Tree: Example

See Example 1.48 and Figure 1.12 on page 70.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Problem

What happens to formulas of the kind $\neg(\phi_1 \wedge \phi_2)$?

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

A Cubic Solver: Idea

Improve the linear solver as follows:

A Cubic Solver: Idea

Improve the linear solver as follows:

- Run linear solver

A Cubic Solver: Idea

Improve the linear solver as follows:

- Run linear solver
- For every node n that is still unmarked:

A Cubic Solver: Idea

Improve the linear solver as follows:

- Run linear solver
- For every node n that is still unmarked:
  - Mark n with `T` and run linear solver, possibly resulting in temporary marks.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

A Cubic Solver: Idea

Improve the linear solver as follows:

- Run linear solver
- For every node n that is still unmarked:
  - Mark n with `T` and run linear solver, possibly resulting in temporary marks.
  - Mark n with `F` and run linear solver, possibly resulting in temporary marks.

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

A Cubic Solver: Idea

Improve the linear solver as follows:

- Run linear solver
- For every node n that is still unmarked:
  - Mark n with `T` and run linear solver, possibly resulting in temporary marks.
  - Mark n with `F` and run linear solver, possibly resulting in temporary marks.
  - Combine temporary marks, resulting in possibly new permanent marks

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Contents

An application of SAT solving: Solve Sudoku Boolean Formula

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

An application of SAT solving: Solve Sudoku Boolean Formula

At the end of Chapter 0, we saw that

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

An application of SAT solving: Solve Sudoku Boolean Formula

At the end of Chapter 0, we saw that

$$\phi = I \wedge R \wedge C \wedge B$$

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

An application of SAT solving: Solve Sudoku Boolean Formula

At the end of Chapter 0, we saw that

$$\phi = I \wedge R \wedge C \wedge B$$

- Note that ϕ is in CNF.

Nguyen An Khuong,
Le Hong Trang,
Huynh Tuong Nguyen,
Tran Van Hoai

An application of SAT solving: Solve Sudoku Boolean Formula

At the end of Chapter 0, we saw that

$$\phi = I \wedge R \wedge C \wedge B$$

- Note that ϕ is in CNF.
- ϕ can be altered so that it contains exactly 3 literals per clause (can be fed to 3-SAT solver).

An application of SAT solving: Solve Sudoku Boolean Formula

At the end of Chapter 0, we saw that

$$\phi = I \wedge R \wedge C \wedge B$$

- Note that ϕ is in CNF.
- ϕ can be altered so that it contains exactly 3 literals per clause (can be fed to 3-SAT solver).
- Problem: Solve this 3-SAT problem with a suitable solver?

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai

Homeworks and Next Week Plan?

Homeworks

- Read carefully all proofs in this note.
- Try to solve the Sudoku in the Introduction note
- Show that $kSAT \in NPC$ for all $k \geq 3$.
- Do ALL marked questions of Exercises 1.6 in [2].
- Read carefully Subsections 1.6.1 and 1.6.2 in [2].

Contents

Homeworks and Next Week Plan?

Homeworks

- Read carefully all proofs in this note.
- Try to solve the Sudoku in the Introduction note
- Show that $kSAT \in NPC$ for all $k \geq 3$.
- Do ALL marked questions of Exercises 1.6 in [2].
- Read carefully Subsections 1.6.1 and 1.6.2 in [2].

Next Week?

Predicate Logic

Nguyen An Khuong, Le Hong Trang, Huynh Tuong Nguyen, Tran Van Hoai