Introduction

Emina Torlak
emina@cs.washington.edu
Today

What is this course about?

Course logistics

Review of propositional logic

A basic SAT solver!
Tools for building better software, more easily
Tools for building better software, more easily more reliable, efficient, secure
Tools for building better software, more easily

automatic verification, debugging & synthesis
Tools for building better software, more easily

```java
class List {
    Node head;

    void reverse() {
        Node near = head;
        Node mid = near.next;
        Node far = mid.next;

        near.next = far;
        while (far != null) {
            mid.next = near;
            near = mid;
            mid = far;
            far = far.next;
        }

        mid.next = near;
        head = mid;
    }
}

class Node {
    Node next; String data;
}
```

Is this list reversal procedure correct?
class List {
    Node head;

    void reverse() {
        Node near = head;
        Node mid = near.next;
        Node far = mid.next;

        near.next = far;
        while (far != null) {
            mid.next = near;
            near = mid;
            mid = far;
            far = far.next;
        }

        mid.next = near;
        head = mid;
    }
}

class Node {
    Node next; String data;
}
class List {
    Node head;

    void reverse() {
        Node near = head;
        Node mid = near.next;
        Node far = mid.next;

        near.next = far;
        while (far != null) {
            mid.next = near;
            near = mid;
            mid = far;
            far = far.next;
        }
        mid.next = near;
        head = mid;
    }
}

class Node {
    Node next; String data;
}
Tools for building better software, more easily

class List {
    Node head;

    void reverse() {
        Node near = head;
        Node mid = near.next;
        Node far = mid.next;

        near.next = far;
        while (far != null) {
            mid.next = near;
            near = mid;
            mid = far;
            far = far.next;
        }

        mid.next = near;
        head = mid;
    }
}

class Node {
    Node next; String data;
}
class List {
    Node head;

    void reverse() {
        Node near = head;
        Node mid = near.next;
        Node far = mid.next;

        near.next = ??;
        while (far != null) {
            mid.next = near;
            near = mid;
            mid = far;
            far = far.next;
        }

        mid.next = near;
        head = mid;
    }
}

class Node {
    Node next; String data;
}
Tools for building better software, more easily

class List {
    Node head;

    void reverse() {
        Node near = head;
        Node mid = near.next;
        Node far = mid.next;
        near.next = null;
        while (far != null) {
            mid.next = near;
            near = mid;
            mid = far;
            far = far.next;
        }
        mid.next = near;
        head = mid;
    }
}

class Node {
    Node next; String data;
}
By the end of this course, you’ll be able to build computer-aided tools for any domain!
By the end of this course, you’ll be able to build computer-aided tools for any domain!
logistics

Topics, structure, people
Course overview

program question

logic

automated reasoning engine

tool
Course overview

program question

verifier, synthesizer, fault localizer

logic

SAT, SMT, model finders
Course overview

**program** | **question**
---|---

**verifier,**  
**synthesizer,**  
**fault localizer**

**logic**

**SAT, SMT, model finders**

---

Fig. 1. Decision procedures can be rather complex... those that we consider in this book take formulas of different theories as input, possibly mix them (using the Nelson–Oppen procedure – see Chap. 10), decide their satisfiability ("YES" or "NO"), and, if yes, provide a satisfying assignment.

Which Theories? Which Algorithms?

A first-order theory can be considered "interesting", at least from a practical perspective, if it fulfills at least these two conditions:

1. The theory is expressive enough to model a real decision problem. Moreover, it is more expressive or more natural for the purpose of expressing some models in comparison with theories that are easier to decide.

---

Drawing from “Decision Procedures” by Kroening & Strichman
Course overview

program question

verifier, synthesizer, fault localizer

logic

SAT, SMT, model finders

Drawing from “Decision Procedures” by Kroening & Strichman
Course overview

logic

SAT, SMT, model finders

program question

verifier, synthesizer, fault localizer

build! (part II)

study (part I)

Fig. 1. Decision procedures can be rather complex... those that we consider in this book take formulas of different theories as input, possibly mix them (using the Nelson–Oppen procedure – see Chap. 10), decide their satisfiability (“YES” or “NO”), and, if yes, provide a satisfying assignment.

Which Theories? Which Algorithms?

A first-order theory can be considered “interesting,” at least from a practical perspective, if it fulfills at least these two conditions:

1. The theory is expressive enough to model a real decision problem. Moreover, it is more expressive or more natural for the purpose of expressing some models in comparison with theories that are easier to decide.

andreis@uw.edu

Drawing from “Decision Procedures” by Kroening & Strichman
Grading

3 individual homework assignments (75%)

- conceptual problems & proofs (TeX)
- implementations (Racket)
- completed on your own (may discuss HWs with course staff only)

Course project (25%)

- build a computer-aided reasoning tool for a domain of your choice
- teams of 2-3 people
- see the course web page for timeline, deliverables and other details
Reading and references

Required readings posted on the course web page

- Complete each reading before the lecture for which it is assigned
- If multiple papers are listed, only the first is required reading

Recommended text books

- Bradley & Manna, *The Calculus of Computation*
- Kroening & Strichman, *Decision Procedures*

Related courses

Advice for doing well in 507

Come to class (prepared)

- Lecture slides are enough to teach from, but not enough to learn from

Participate

- Ask and answer questions

Meet deadlines

- Turn homework in on time
- Start homework and project sooner than you think you need to
- Follow instructions for submitting code (we have to be able to run it)
- No proof should be longer than a page (most are ~1 paragraph)
People

Emina Torlak
PLSE
CSE 596
Wed 2-3pm

Eric Butler
Game Science & PLSE
CSE 324
Thu 2-3pm
People

Instructor

Emina Torlak
PLSE
CSE 596
Thursdays 9-10

TA

Eric Butler
Game Science & PLSE
CSE 324
Thu 2-3pm

Students!

Your name
Research area
Let’s get started! A review of propositional logic

- Syntax
- Semantics
- Satisfiability and validity
- Proof methods
- Semantic judgments
- Normal forms (NNF, DNF, CNF)
Syntax of propositional logic

\((\neg p \land T) \lor (q \rightarrow \bot)\)
Syntax of propositional logic

Atom

truth symbols: $\top$ ("true"), $\bot$ ("false")

propositional variables: $p, q, r, \ldots$

Formula

$a$ literal or the application of a logical connective

$\neg \circ F, F_1, F_2$:

¬ $\text{"not"}$ (negation)

"and" (conjunction)

"or" (disjunction)

"implies" (implication)

"if and only if" (iff)
Syntax of propositional logic

Atom

**truth symbols**: \( \top \) ("true"), \( \bot \) ("false")

**propositional variables**: \( p, q, r, \ldots \)

Literal

an atom \( \alpha \) or its negation \( \neg \alpha \)

\[
(\neg p \land \top) \lor (q \rightarrow \bot)
\]
Syntax of propositional logic

Atom

truth symbols: \( \top \) (“true”), \( \bot \) (“false”)
propositional variables: \( p, q, r, \ldots \)

Literal

an atom \( \alpha \) or its negation \( \neg \alpha \)

Formula

a literal or the application of a logical connective to formulas \( F, F_1, F_2 \):

\[
\neg F \quad \text{“not”} \quad \text{(negation)}
\]

\[
F_1 \land F_2 \quad \text{“and”} \quad \text{(conjunction)}
\]

\[
F_1 \lor F_2 \quad \text{“or”} \quad \text{(disjunction)}
\]

\[
F_1 \rightarrow F_2 \quad \text{“implies”} \quad \text{(implication)}
\]

\[
F_1 \leftrightarrow F_2 \quad \text{“if and only if”} \quad \text{(iff)}
\]

\[ (-\neg p \land \top) \lor (q \rightarrow \bot) \]
A **interpretation** \( I \) for a propositional formula \( F \) maps every variable in \( F \) to a truth value:

\[
I : \{ \ p \mapsto \text{true}, \ q \mapsto \text{false}, \ldots \}
\]
An **interpretation** $I$ for a propositional formula $F$ maps every variable in $F$ to a truth value:

$$I : \{ p \mapsto \text{true}, q \mapsto \text{false}, \ldots \}$$

$I$ is a **satisfying interpretation** of $F$, written as $I \models F$, if $F$ evaluates to true under $I$.

$I$ is a **falsifying interpretation** of $F$, written as $I \not\models F$, if $F$ evaluates to false under $I$. 
Semantics of propositional logic: definition

Base cases:

• $I \vDash \top$
• $I \nvDash \bot$
• $I \vDash p$ iff $I[p] = \text{true}$
• $I \nvDash p$ iff $I[p] = \text{false}$
Semantics of propositional logic: definition

Base cases:
- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \text{true}$
- $I \not\models p$ iff $I[p] = \text{false}$

Inductive cases:
Semantics of propositional logic: definition

**Base cases:**

- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \text{true}$
- $I \not\models p$ iff $I[p] = \text{false}$

**Inductive cases:**

- $I \models \neg F$ iff $I \not\models F$
Semantics of propositional logic: definition

**Base cases:**
- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \text{true}$
- $I \not\models p$ iff $I[p] = \text{false}$

**Inductive cases:**
- $I \models \neg F$ iff $I \not\models F$
- $I \models F_1 \land F_2$ iff $I \models F_1$ and $I \models F_2$
Semantics of propositional logic: definition

**Base cases:**
- \( I \vdash \top \)
- \( I \nvDash \bot \)
- \( I \vdash p \) iff \( I[p] = \text{true} \)
- \( I \nvDash p \) iff \( I[p] = \text{false} \)

**Inductive cases:**
- \( I \vdash \neg F \) iff \( I \nvDash F \)
- \( I \vdash F_1 \land F_2 \) iff \( I \vdash F_1 \) and \( I \vdash F_2 \)
- \( I \vdash F_1 \lor F_2 \) iff \( I \vdash F_1 \) or \( I \vdash F_2 \)
- \( I \vdash F_1 \rightarrow F_2 \) iff \( I \nvDash F_1 \) or \( I \vdash F_2 \)
- \( I \vdash F_1 \leftrightarrow F_2 \) iff \( I \vdash F_1 \) and \( I \vdash F_2 \), or \( I \nvDash F_1 \) and \( I \nvDash F_2 \)
Semantics of propositional logic: example

\[ F: \quad (p \land q) \rightarrow (p \lor \neg q) \]

\[ l: \quad \{ p \leftrightarrow \text{true}, q \leftrightarrow \text{false} \} \]
Semantics of propositional logic: example

\[ F: \ (p \land q) \rightarrow (p \lor \neg q) \]
\[ I: \ \{p \mapsto \text{true}, \ q \mapsto \text{false}\} \]
\[ I \vDash F \]
Satisfiability & validity of propositional formulas

\( F \) is **satisfiable** iff \( I \models F \) for some \( I \).

\( F \) is **valid** iff \( I \models F \) for all \( I \).
Satisfiability & validity of propositional formulas

- $F$ is **satisfiable** iff $I \models F$ for some $I$.
- $F$ is **valid** iff $I \models F$ for all $I$.

**Duality** of satisfiability and validity:

- $F$ is valid iff $\neg F$ is unsatisfiable.
Satisfiability & validity of propositional formulas

\[ F \text{ is satisfiable iff } I \models F \text{ for some } I. \]

\[ F \text{ is valid iff } I \models F \text{ for all } I. \]

**Duality** of satisfiability and validity:

\[ F \text{ is valid iff } \neg F \text{ is unsatisfiable.} \]

If we have a procedure for checking satisfiability, then we can also check validity of propositional formulas, and vice versa.
Techniques for deciding satisfiability & validity

Search

Deduction

SAT solver
Enumerate all interpretations (i.e., build a truth table), and check that they satisfy the formula.
Enumerate all interpretations (i.e., build a truth table), and check that they satisfy the formula.

Assume the formula is invalid, apply proof rules, and check for contradiction in every branch of the proof tree.

SAT solver
Proof by search (truth tables)

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \land q$</th>
<th>$\neg q$</th>
<th>$p \lor \neg q$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Proof by search (truth tables)

\[ F: \ (p \land q) \rightarrow (p \lor \neg q) \]

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Valid.
Proof by deduction (semantic arguments)

Example proof rules:

\[
\begin{align*}
I \models \neg F & \quad I \vdash F_1 \land F_2 \\
I \not\models F & \quad I \models F_1 \\
& \quad I \models F_2 \\
I \not\models \neg F & \quad I \not\models F_1 \land F_2 \\
I \models F & \quad I \not\models F_1 \quad I \not\models F_2
\end{align*}
\]
Proof by deduction (semantic arguments)

Example proof rules:

\[
egin{align*}
I &
\models \neg F &
I &
\models F_1 \land F_2 \\
I &
\not\models F &
I &
\models F_1 \\
& &
I &
\models F_2 \\
I &
\not\models \neg F &
I &
\not\models F_1 \land F_2 \\
& &
I &
\not\models F_1 \quad I &
\not\models F_2 \\
I &
\models F &
I &
\models F_2 \\
\end{align*}
\]

\[F: p \land \neg q\]
Proof by deduction (semantic arguments)

Example proof rules:

\[ \frac{I \models \neg F}{I \not\models F} \quad \frac{I \models F_1 \land F_2}{\begin{array}{l} I \models F_1 \\ I \models F_2 \end{array}} \]

\[ \frac{I \not\models \neg F}{I \models F} \quad \frac{I \not\models F_1 \land F_2}{\begin{array}{l} I \not\models F_1 \\ I \not\models F_2 \end{array}} \]

\[ F: \ p \land \neg q \]

1. \( I \not\models p \land \neg q \) (assumption)
Proof by deduction (semantic arguments)

Example proof rules:

\[
\begin{align*}
\frac{I \models \neg F}{I \not\models F} & \quad \frac{I \not\models F_1 \land F_2}{I \models F_1} \\
& \quad \frac{I \models F_2}{I \not\models F} & \quad \frac{I \not\models F_1 \land F_2}{I \not\models F_1} \\
& \quad \frac{I \not\models F_2}{I \models F} \end{align*}
\]

\[F: \ p \land \neg q\]

1. \(I \not\models p \land \neg q\) (assumption)
   a. \(I \not\models p\) (1, \(\land\))
Proof by deduction (semantic arguments)

Example proof rules:

\[
\begin{array}{c}
I \models \neg F \\
\hline
I \not\models F
\end{array}
\]

\[
\begin{array}{c}
I \models F_1 \land F_2 \\
\hline
I \not\models F_1 \\
I \not\models F_2
\end{array}
\]

\[
\begin{array}{c}
I \not\models \neg F \\
\hline
I \models F
\end{array}
\]

\[
\begin{array}{c}
I \not\models F_1 \land F_2 \\
\hline
I \not\models F_1 \\
I \not\models F_2
\end{array}
\]

F: \quad p \land \neg q

1. I \not\models p \land \neg q \quad \text{(assumption)}
   a. I \not\models p \quad (1, \land)
   b. I \not\models \neg q \quad (1, \land)
Proof by deduction (semantic arguments)

Example proof rules:

\[ \frac{I \models \neg F}{I \not\models F} \]

\[ \frac{I \not\models F_1 \land F_2}{I \not\models F_1} \quad \frac{I \not\models F_1 \land F_2}{I \not\models F_2} \]

\[ \frac{I \models \neg \neg F}{I \models F} \]

\[ F: \quad p \land \neg q \]

1. \( I \not\models p \land \neg q \) (assumption)
   a. \( I \not\models p \) (1, \( \land \))
   b. \( I \not\models \neg q \) (1, \( \land \))
   i. \( I \models q \) (1b, \( \neg \))
Proof by deduction (semantic arguments)

Example proof rules:

\[ \frac{I \vdash \neg F}{I \not\models F} \]

\[ \frac{I \vdash F_1 \land F_2}{I \vdash F_1} \]

\[ I \vdash F_2 \]

\[ \frac{I \not\models \neg F}{I \not\models F_1 \land F_2} \]

\[ \frac{I \not\models F_1 \land F_2}{I \not\models F_1} \]

\[ I \not\models F_2 \]

\[ F: p \land \neg q \]

1. \( I \not\models p \land \neg q \) (assumption)
   a. \( I \not\models p \) (1, \land)
   b. \( I \not\models \neg q \) (1, \land)
   i. \( I \models q \) (1b, \neg)

Invalid; \( I \) is a falsifying interpretation.
Semantic judgements

Formulas $F_1$ and $F_2$ are equivalent, written $F_1 \iff F_2$, iff $F_1 \leftrightarrow F_2$ is valid.

Formula $F_1$ implies $F_2$, written $F_1 \implies F_2$, iff $F_1 \rightarrow F_2$ is valid.

$F_1 \iff F_2$ and $F_1 \implies F_2$ are not propositional formulas (not part of syntax). They are properties of formulas, just like validity or satisfiability.
Semantic judgements

Formulas $F_1$ and $F_2$ are **equivalent**, written $F_1 \iff F_2$, iff $F_1 \leftrightarrow F_2$ is valid.

Formula $F_1$ **implies** $F_2$, written $F_1 \implies F_2$, iff $F_1 \rightarrow F_2$ is valid.

If we have a procedure for checking satisfiability, then we can also check for equivalence and implication of propositional formulas.
Semantic judgements

Formulas $F_1$ and $F_2$ are **equivalent**, written $F_1 \iff F_2$, iff $F_1 \leftrightarrow F_2$ is valid.

Formula $F_1$ **implies** $F_2$, written $F_1 \implies F_2$, iff $F_1 \rightarrow F_2$ is valid.

If we have a procedure for checking satisfiability, then we can also check for equivalence and implication of propositional formulas.

Why do we care?
Getting ready for SAT solving with normal forms

A normal form for a logic is a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form.
A **normal form** for a logic is a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form.

Assembly language for a logic.
A **normal form** for a logic is a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form.

Three important normal forms for propositional logic:
- Negation Normal Form (NNF)
- Disjunctive Normal Form (DNF)
- Conjunctive Normal Form (CNF)

Assembly language for a logic.
Negation Normal Form (NNF)

Atom := Variable | ⊤ | ⊥
Literal := Atom | ¬Atom
Formula := Literal | Formula op Formula
op := ∧ | ∨
Negation Normal Form (NNF)

Atom := Variable | ⊤ | ⊥
Literal := Atom | ¬Atom
Formula := Literal | Formula op Formula
op := ∧ | ∨

• The only allowed connectives are ∧, ∨, and ¬.
• ¬ can appear only in literals.
Negation Normal Form (NNF)

Atom := Variable | T | ⊥
Literal := Atom | ¬Atom
Formula := Literal | Formula op Formula
op := ∧ | ∨

• The only allowed connectives are ∧, ∨, and ¬.
• ¬ can appear only in literals.

Conversion to NNF performed using **DeMorgan’s Laws**:

\[ \neg(F \land G) \iff \neg F \lor \neg G \]
\[ \neg(F \lor G) \iff \neg F \land \neg G \]
Disjunctive Normal Form (DNF)

Atom := Variable | ⊤ | ⊥
Literal := Atom | ¬Atom
Formula := Clause ∨ Formula
Clause := Literal | Literal ∧ Clause
Disjunctive Normal Form (DNF)

Atom := Variable | \( \top \) | \( \bot \)
Literal := Atom | \( \neg \)Atom
Formula := Clause \( \lor \) Formula
Clause := Literal | Literal \( \land \) Clause

- Disjunction of conjunction of literals.
Disjunctive Normal Form (DNF)

Atom := Variable | $\top$ | $\bot$
Literal := Atom | $\neg$Atom
Formula := Clause $\lor$ Formula
Clause := Literal | Literal $\land$ Clause

- Disjunction of conjunction of literals.

To convert to DNF, convert to NNF and distribute $\land$ over $\lor$:

\[(F \land (G \lor H)) \iff (F \land G) \lor (F \land H)\]
\[((G \lor H) \land F) \iff (G \land F) \lor (H \land F)\]
Disjunctive Normal Form (DNF)

Atom := Variable | \( \top \) | \( \bot \)
Literal := Atom | \( \neg \)Atom
Formula := Clause \( \lor \) Formula
Clause := Literal | Literal \( \land \) Clause

- Disjunction of conjunction of literals.
- Deciding satisfiability of a DNF formula is trivial.

To convert to DNF, convert to NNF and distribute \( \land \) over \( \lor \):

- \((F \land (G \lor H)) \iff (F \land G) \lor (F \land H)\)
- \(((G \lor H) \land F) \iff (G \land F) \lor (H \land F)\)
Disjunctive Normal Form (DNF)

Atom := Variable | $\top$ | $\bot$
Literal := Atom | $\neg$Atom
Formula := Clause $\lor$ Formula
Clause := Literal | Literal $\land$ Clause

- Disjunction of conjunction of literals.
- Deciding satisfiability of a DNF formula is trivial.
- Why not SAT solve by conversion to DNF?

To convert to DNF, convert to NNF and distribute $\land$ over $\lor$:

$$(F \land (G \lor H)) \iff (F \land G) \lor (F \land H)$$

$$( (G \lor H) \land F ) \iff (G \land F) \lor (H \land F)$$
Conjunctive Normal Form (CNF)

Atom := Variable | $\top$ | $\bot$
Literal := Atom | $\neg$Atom
Formula := Clause $\land$ Formula
Clause := Literal | Literal $\lor$ Clause

- Conjunction of disjunction of literals.
- Deciding the satisfiability of a CNF formula is hard.
- SAT solvers use CNF as their input language.

To convert to CNF, convert to NNF and distribute $\lor$ over $\land$

\[(F \lor (G \land H)) \iff (F \lor G) \land (F \lor H)\]
\[((G \land H) \lor F) \iff (G \lor F) \land (H \lor F)\]
Conjunctive Normal Form (CNF)

Atom := Variable | T | ⊥
Literal := Atom | ¬ Atom
Formula := Clause ∧ Formula
Clause := Literal | Literal ∨ Clause

• Conjunction of disjunction of literals.
• Deciding the satisfiability of a CNF formula is hard.
• SAT solvers use CNF as their input language.

Why CNF? Doesn’t the conversion explode just as badly as DNF?

To convert to CNF, convert to NNF and distribute ∨ over ∧

(F ∨ (G ∧ H)) ⇔ (F ∨ G) ∧ (F ∨ H)
((G ∧ H) ∨ F) ⇔ (G ∨ F) ∧ (H ∨ F)
Equisatisfiability and Tseitin’s transformation
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

**Key idea:** introduce **auxiliary variables** to represent the output of subformulas, and constrain these variables using CNF clauses.
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce auxiliary variables to represent the output of subformulas, and constrain those variables using CNF clauses.
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

\[
\begin{align*}
x & \rightarrow (y \land z) \\
a_1 & \rightarrow (x \rightarrow a_2) \\
(x \rightarrow a_2) & \rightarrow a_1 \\
a_2 & \leftrightarrow (y \land z)
\end{align*}
\]
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.
Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

**Tseitin’s transformation** converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

\[
\begin{align*}
    &x \rightarrow (y \land z) \\
    &a1 \\
    &\neg a1 \lor \neg x \lor a2 \\
    &x \lor a1 \\
    &\neg a2 \lor a1 \\
    &a2 \leftrightarrow (y \land z)
\end{align*}
\]
A basic SAT solver!
// Returns true if the CNF formula F is satisfiable; otherwise returns false.

DPLL(F)
    G ← BCP(F)
    if G = ⊤ then return true
    if G = ⊥ then return false
    p ← choose(vars(G))
    return DPLL(G{p ↦ ⊤}) ||
           DPLL(G{p ↦ ⊥})
// Returns true if the CNF formula F is satisfiable; otherwise returns false.

DPLL(F)
    G ← BCP(F)
    if G = √ then return true
    if G = ⊥ then return false
    p ← choose(vars(G))
    return DPLL(G{p ↦ √}) ||
                  DPLL(G{p ↦ ⊥})

Boolean constraint propagation applies unit resolution until fixed point:

\[
\text{lit clause}[\text{lit}] \\
\text{lit clause}[-\text{lit}] \\
\text{clause}[\bot]
\]
Summary

Today

• Course overview & logistics
• Review of propositional logic
• A basic SAT solver

Next Lecture

• A modern SAT solver
• Read Chapter 1 of Bradley & Manna