Propositional Logic III: SAT Solvers

Type Theory and Mechanized Reasoning

Lecture 7

CAS CS 400

Introduction

Administrivia

Homework 2 is due on Thursday 11:59PM. Homework 1 is due ASAP.

We'll talk about the final project briefly on Wednesday.

We're going to try something new with our standard library. Thanks for your patience.

Objectives

Finish discussing semantics notions in propositional logic.

Define conjunctive normal forms (CNFs)

Start discussing SAT solvers and the DPLL procedure.

Agda Tutorial: CS400-Lib

Setting up a Library

1. Clone the course library repo somewhere on your machine.

2. Include the library file in your Agda **libraries** file.

3. Include the library name in your Agda **defaults** file.

Example

```
reverse : {A : Set} -> List A -> List A
reverse {A} = go {A} [] where
  go : {A : Set} -> List A -> List A -> List A
  go acc [] = acc
  go acc (x :: xs) = go (x :: acc) xs
```

No unicode.

CS400-Lib is easier to read through than the standard library.

Recap: Semantic Notions

Validity

Definition. A formula ϕ is **valid** if every valuation makes it true.

That is, $\overline{v}(\phi) = \mathsf{true}$ for any valuation v.

Example. $\neg(x \vee y) \rightarrow \neg x \wedge \neg y$

Satisfiability

Definition. A formula ϕ is **satisfiable** if there is *some* valuation which makes ϕ true.

That is, $\overline{v}(\phi) = \mathsf{true}$ for some valuation v.

<u>Example.</u> $(x \vee y) \wedge (x \vee \neg y)$

Entailment

Definition. A set of formulas $\Gamma = \{\psi_1, \ldots, \psi_n\}$ **entails** a formula ϕ if every valuation which makes every formula in Γ true also make ϕ true.

That is, if $\overline{v}(\psi_1) = \ldots = \overline{v}(\psi_n) = \text{true}$ then $\overline{v}(\phi) = \text{true}$.

Example. $\{\neg(x \vee y)\} \vDash \neg x \wedge \neg y$

Two Key Meta-Theoretic Results

Deduction Theorem. $\Gamma \cup \{\psi\} \vDash \phi$ if and only if $\Gamma \vDash \psi \to \phi$.

Example. $\vDash \neg(x \vee y) \to (\neg x \vee \neg y)$ iff $\neg(x \vee y) \vDash \neg x \wedge \neg y$.

Validity/Unsatisfiability Theorem. ϕ is a tautology if and only if $\neg\phi$ is unsatisfiable.

Example. If we want to show $\psi \vDash \phi$, it suffices to show that $\neg(\psi \to \phi)$ is unsatisfiable.

Logical Equivalence

Definition. Formulas ϕ and ψ are **logically equivalent** if $\phi \vDash \psi$ and $\psi \vDash \phi$.

That is, $\overline{v}(\phi) = \overline{v}(\psi)$ for any valuation v.

Example. $\neg(x \vee y) \equiv \neg x \wedge \neg y$

DeMorgan's Law

$$P \lor Q \equiv \neg(\neg P \land \neg Q)$$
$$P \land Q \equiv \neg(\neg P \lor \neg Q)$$

The Takeaway. We can write a disjunction (or) in terms of negation (not) and conjunction (and).

Exclusive Disjunction

$$P \oplus Q$$

$P \oplus Q$ stands for "exactly one of P and Q is true".

Why didn't we include this in our notion of logic?

Theorem. For any formulas P and Q

$$P \oplus Q \equiv (\neg P \wedge Q) \vee (P \wedge \neg Q)$$

Boolean Functions

Definition. An n–variate **Boolean function** is a function of the form

$$F : \{T, F\}^n \to \{T, F\}$$

Example.

$$\text{XOR}(F, F) = F$$
$$\text{XOR}(T, F) = T$$
$$\text{XOR}(F, T) = T$$
$$\text{XOR}(T, T) = F$$

Functional Completeness

Definition. An n–variate Boolean function is **represented** by a formula ϕ over variables $x_1, \ldots, x_n$ if, for any valuation v

$$\bar{v}(\phi) = F(v(x_1), \ldots, v(x_n))$$

Theorem. Every Boolean formula is represented by a propositional formula.

Complete Sets of Connectives

Definition. A set of connectives is **complete** if every Boolean function can be represented by this set of connectives.

Theorem. $\{\neg, \vee\}$ and $\{\neg, \wedge\}$ are complete sets of connectives.

Understanding Check

Show that $\{\text{NAND}\}$ *is a complete set of connectives.*

Show that $\{\rightarrow\}$ *is not a complete set of connectives.*

Normal Forms

Motivation

It is more useful computational to have a formula in a simple form.

There are many normal forms for formulas, but we will consider one primary form: **Conjunctive Normal Form (CNF)**, e.g.

literals $$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_1 \vee x_4) \wedge (x_4 \wedge \neg x_5)$$

clause

Conjunctive Normal Form (CNF)

A **literal** is a variable or its negation:

$$x, \neg y, \dots$$

A **clause** is a disjunction of literals:

$$x \vee \neg y \vee z \vee \neg w$$

A **conjunctive normal form (CNF)** formula is a conjunction of clauses:

$$(x \vee \neg y) \wedge (y \vee \neg z) \vee (z \vee w \vee \neg y)$$

Literal Notation

$$x \Longrightarrow x^0$$

$$\neg x \Longrightarrow x^1$$

It will be convenient to use the following notation for literals.

Example. x^{1-a} is logically equivalent to $\neg x^a$

CNFs in Agda

new library function

```
Literal : Set
Literal = Nat & Bool

Clause : Set
Clause = List Literal

CNF : Set
CNF = List Clause
```

The simplicity of representation makes algorithms easier to write.

The Key Meta-Theoretic Result

Theorem. Every formula is logically equivalent to a CNF formula.

This reduces the problem of determining validity or entailment to determining the satisfiability of a CNF formula.

SAT Solvers

SAT

Satisfiability of CNF formulas (SAT) is a fundamental problem in complexity theory.

Theorem (Cook, Levin). SAT is NP–complete.

If we could solve SAT in polynomial time, then we could solve a lot of hard computational problems in polynomial time.

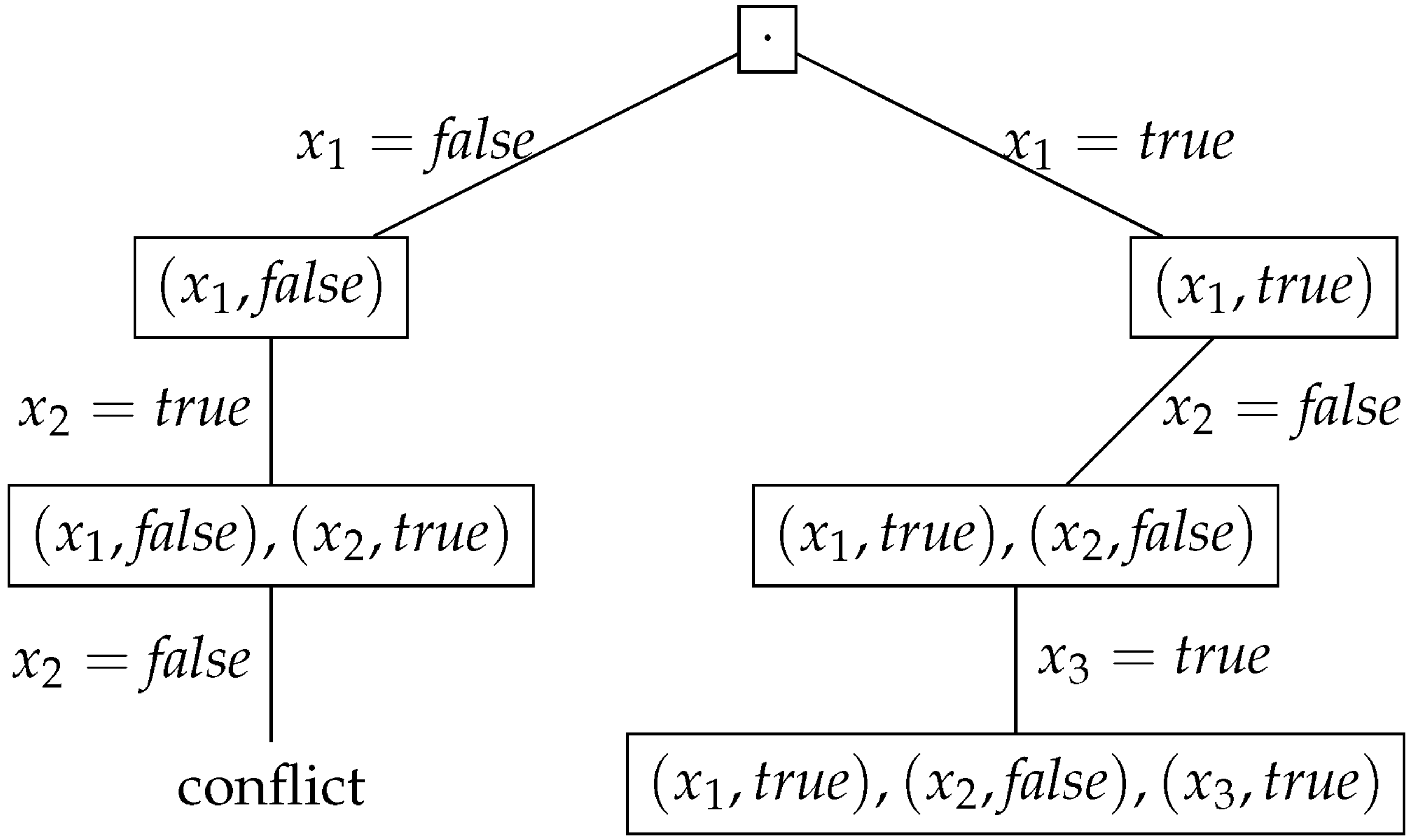
SAT Solvers

There are people building *powerful* algorithms for SAT and using them to solve real world problems.

(Since a lot of hard problems reduce to SAT, people even use these algorithms as NP-oracles.)

What do these algorithms look like?

A Simple Algorithm: DPLL



Idea. Build a satisfying assignment one variable at a time, updating the formula at each step.

This is a backtracking procedure, where a leaf is a satisfying assignment or a **conflict** (the assignment cannot be satisfying).

https://www.mdpi.com/2227-7390/10/13/2264

Partial Assignments

Definition. A **partial assignment** is a set of literals.

Example. $\{x^1, y^0, z^1\}$

We think of this as a set of *assertions*, i.e., x is true and y is false and z is true.

Restriction by a Literal

Definition. Given a formula ϕ and a literal l the **restriction** of ϕ by l, written $\phi|_l$ is given by

$$C|_l = C \text{ if } l \notin C$$

$$(C \lor x^a \lor D)|_{x^b} = \begin{cases} C \lor D & a \neq b \\ \text{true} & \text{otherwise} \end{cases}$$

$$(C_1 \land C_2 \land \ldots \land C_k)|_l = (C_1|_l) \land (C_2|_l) \ldots \land (C_k|_l)^*$$

*This can be made more efficient.

Example

Let's compute:

$$((x^0 \vee y^1 \vee z^1) \wedge x^0 \wedge (y^0 \vee z^0) \vee (x^1 \vee w^1))\,\big|_{x^1}$$

Restriction by a Literal in Agda

```
restrictC : Literal -> Clause -> Clause
restrictC l [] = []
restrictC l (x :: xs) with eqL l x
restrictC l (x :: xs) | true = trueC
restrictC l (x :: xs) | false with opL l x
restrictC l (x :: xs) | false | true = restrictC l xs
restrictC l (x :: xs) | false | false = x :: restrictC l xs

restrict : Literal -> CNF -> CNF
restrict l f = Lists.map (restrictC l) f
```

eqL and **opL** determine l is equal, or equal but negated.

General Restriction

Restriction by a partial assignment can be understood as repeated restriction by literals, e.g.*

$$\phi|_{\{l_1,l_2\}} = (\phi|_{l_1})|_{l_2}$$

*This elides questions about order of restrictions

Let's try it in Agda.

Naive DPLL in Agda

```
{-# TERMINATING #-}
is-sat : CNF -> Bool
is-sat f with find-var f
is-sat f | Nothing = notb (has-empty f)
is-sat f | Just x with is-sat (restrict (x , true) f)
is-sat f | Just x | true = true
is-sat f | Just x | false = is-sat (restrict (x , false) f)
```

High Level: is-sat branches the choice of restricting by x or $\neg x$.

Heuristics

Unit Propagation. If the formula has a clause which is a single literal l, then restrict the formula by l.

Example. $(x^0 \wedge (x^1 \vee y^0))|_{x^0}$ becomes y^0

Pure Literal Rule. If the formula has only appearance of l, then restrict the formula by l.

Example. $((x^0 \vee y^0) \wedge (x^0 \vee y^1) \wedge z^0|)_{x^0}$ becomes z^0

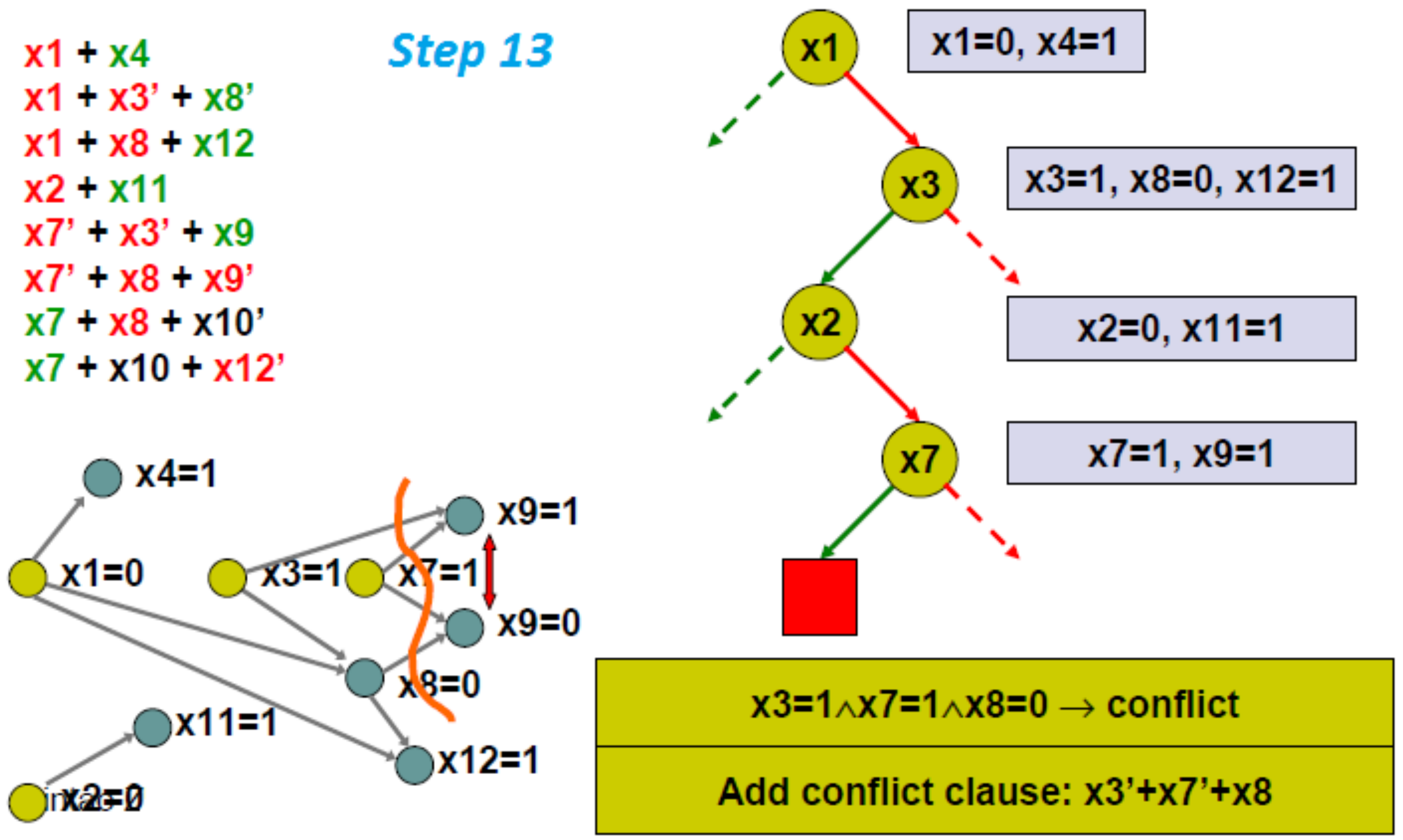
And that's it. That's the David–Putnam–Logemann–Loveland Procedure.

Example

`Let's walk through an example:`

$$
\begin{aligned}
&(x^0 \vee y^1 \vee z^1) \wedge \\
&(x^1 \vee z^1 \vee w^1) \wedge \\
&(x^1 \vee z^1 \vee w^0) \wedge \\
&(x^1 \vee z^0 \vee w^1) \wedge \\
&(x^1 \vee z^0 \vee w^0)
\end{aligned}
$$

A More Complicated Algorithm: CDCL



Modern SAT solvers are built using a heuristic called **conflict driven clause learning (CDCL).**

Idea. When we find that a partial assignment creates a conflict, we can *add* clauses to our formula which might help the solver avoid making the same mistake again.

https://en.wikipedia.org/wiki/Conflict-driven_clause_learning#/media/File:Cdcl13.png