Nelson-Oppen

Specifications section, **Algorithms** topic, **Lecture 6**

Pavel Panchekha

CS 6110, U of Utah

23 January 2020

First-order Logic

First-order logic is **common framework** for logics

$$p, q := \neg p \mid p \vee q \mid p \wedge q \mid \forall v, p \mid \exists v, p$$

Can add **domain-specific** constructs

- **Sorts, constants, functions, and relations** define the syntax
- **Interpretations** of each define the meaning
- **Axioms** allow proving first-order statements

Domains can be **mixed** to solve complex problems

Proofs

Proofs provide **compact evidence** for a first-order fact

$$[\,] \quad \underbrace{A_1, A_2, \ldots, A_n}_{\textbf{Axioms}} \quad \vdash \quad \underbrace{p}_{\textbf{Fact}}$$

Quantifier rules for manipulating facts & axioms

$$\frac{[\Gamma, x]\ A \vdash p}{[\Gamma]\ A \vdash \forall x, p} \qquad \frac{[\Gamma]\ A \vdash p[x := e]}{[\Gamma]\ A \vdash \exists x, p} \qquad \frac{[\Gamma, x]\ A \vdash p}{[\Gamma]\ \exists x, A \vdash p} \qquad \frac{[\Gamma]\ A[x := e] \vdash p}{[\Gamma]\ \forall x, A \vdash p}$$

Replace identical expressions with boolean variables

Theories

Theory of **equality**: reflection and substitution

- Quantifier-free vs quantified formulas

Theory of **arithmetic**: induction and definitions

- Choosing subsets of the theory for easy solving

Theory of **arrays**: building one theory out of others

- Adding operators via definitions

Class Progress

Logical reasoning

Program logics

Static analysis

First-order Logic

Decision Procedures

Mixing Theories

Equality

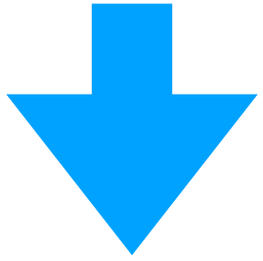
Integers

Arrays

Algorithms

Math

$$\forall x, \forall y, x \times x = y \rightarrow \exists k, \cdots$$



Programs

```
for i in l: recurse(v[i])
```

Mixing Theories

Converting first-order problems to a **standard form**

Combining conjunctive form, prefix form, and satisfiability

Separating problems across different theories

Assigning atomic clauses to theories

Interfacing between theories

Matching assumptions in different domains

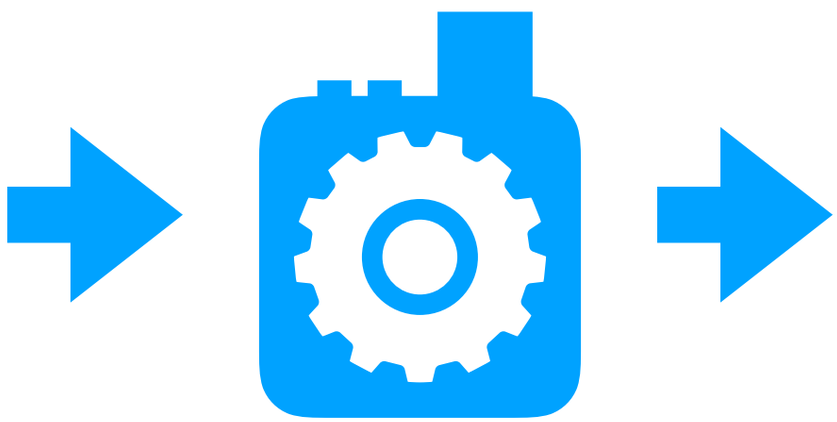
Solver Architecture

A standard prover for first-order logic

The Goal

First-order statement

p



Proof of p

or

Proof of $\neg p$

Impossible in general. Gödel, halting problem, etc…

- **Domain-specific reasoning** is essential
- **Quantifiers instantiation** difficult in general
- **Domains like integers** have complex axiom schemas
- Mosaic of **purpose-built fragments** to reason about

The Problems

Impossible in general. Gödel, halting problem, etc…

Domain-specific reasoning is essential

→ **Embed domain-specific solvers**

Quantifiers instantiation difficult in general

→ **Restrict to quantifier-alternation-free statements**

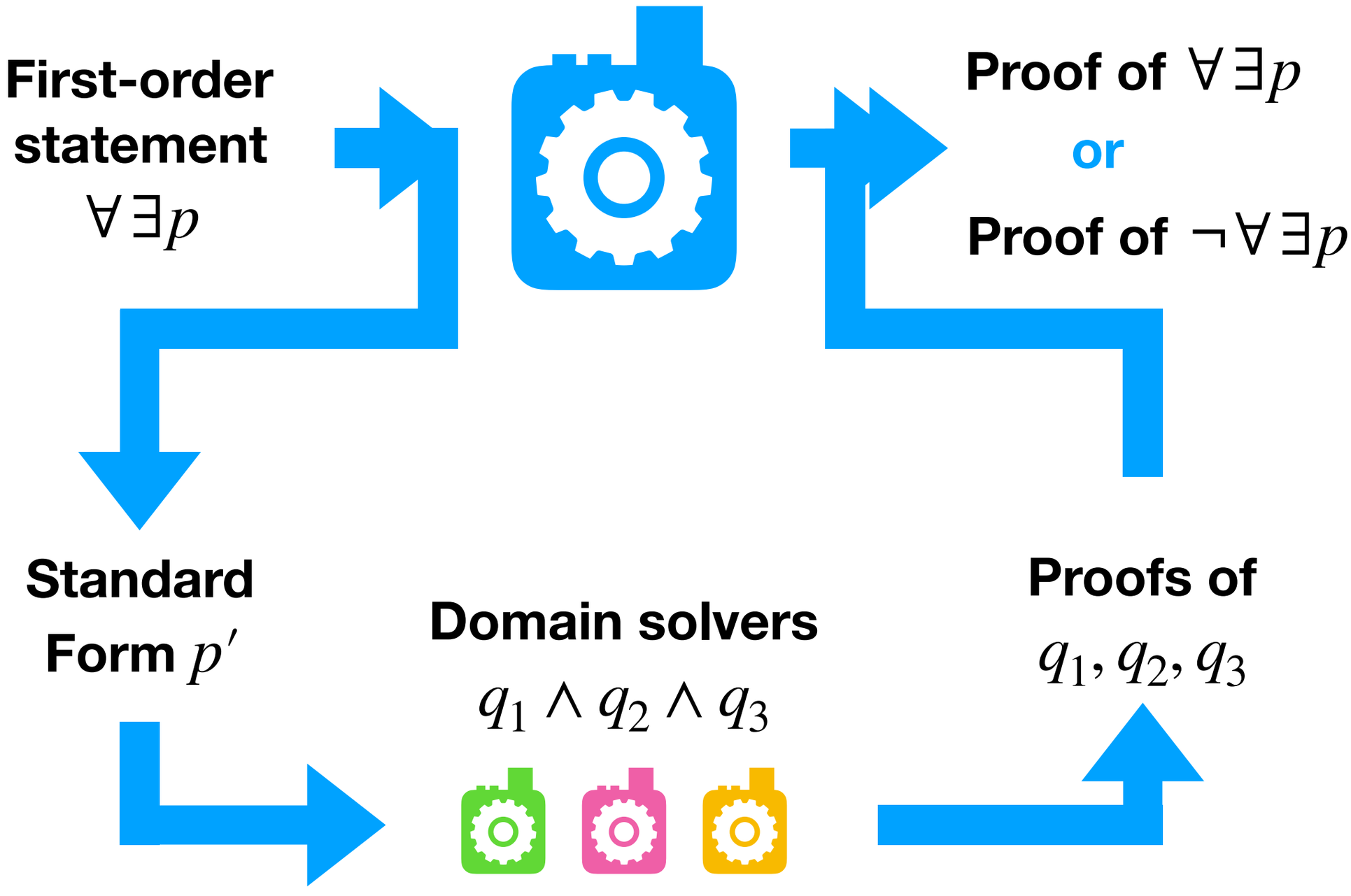
Domains like integers have complex axiom schemas

→ **Custom algorithms for using those axioms**

Mosaic of **purpose-built fragments** to reason about

→ **Split single statement across available fragments**

The Approach



Standard Query Form

$$\forall x, (\exists y, P(x, y)) \land (\forall z, R(x, z) \rightarrow Q(x))$$

Prefix form $\forall x, \exists y, \exists z, P(x, y) \land (R(x, z) \rightarrow Q(x))$

$\forall x, \exists y, \exists z, p$

Heuristics

$\forall x, p[y := e_1, z := e_2]$

$\forall x, \forall y, \forall z, p$

Opposite

$\exists x, \exists y, \exists z, \neg p$

$\exists x, \exists y, \exists z, p$

Standard Form

Satisfying assignment for p

What Next?

Standard Form $\exists x, \exists y, \exists z, p$

Conjunctive Form $(a_1 \vee \neg a_2) \wedge (\neg b_1 \vee \cdots) \wedge \cdots)$

DPLL(T)

Literal Assignment $a_1 \wedge a_2 \wedge \neg b_1 \wedge \cdots$

Nelson-Oppen

$a_1 \wedge \neg b_1 \wedge \cdots$

Solver A

$a_2 \wedge \cdots$

Solver B

DPLL(T)

Simplified domain solver queries

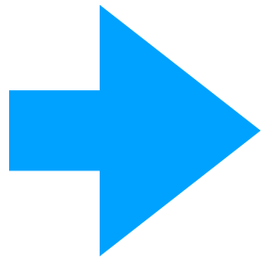
DPLL review

Algorithm finds **satisfying assignment** for boolean logic

Maintains a set of **variable assignments**

Statement p

$$(\neg a_1 \vee a_2) \wedge (\neg b_1 \vee b_2) \wedge \ldots$$



Assignment Γ

$$a_1 \wedge \neg b_2 \wedge \ldots$$

Summary of the DPLL algorithm:

Extend Γ with **guessed or inferred** value until:

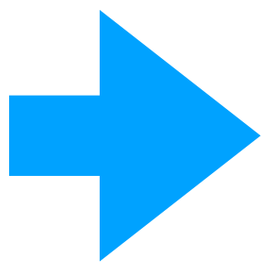
$\Gamma \vdash p$	$\Gamma \vdash \neg p$	Can't extend Γ
SAT	**BACKTRACK**	**UNSAT**

DPLL with Theories

Want **extension to first-order theories**

Statement p

$$(\neg a_1 \vee a_2) \wedge (\neg b_1 \vee b_2) \wedge \ldots$$

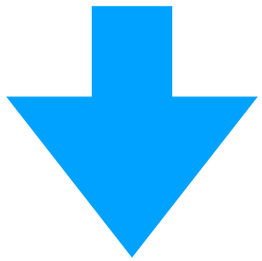


Assignment Γ

$$a_1 \wedge \neg b_2 \wedge \ldots$$

The $a_i, b_i, \ldots$ are **literals from the theory**

$$(\neg(\underbrace{x < 0}_{a_1}) \vee (\underbrace{x \times x > 0}_{a_2})) \wedge (\neg(\underbrace{x = 0}_{b_1}) \vee (\underbrace{x \times x = 0}_{b_2}) \wedge \ldots$$



Γ is impossible

$$\underbrace{x < 0}_{a_1} \wedge (\underbrace{x \times x = 0}_{b_2}) \wedge \ldots$$

DPLL with Theories

Want **extension to first-order theories**

Γ **is impossible** $x < 0 \wedge (x \times x = 0) \wedge \ldots$

What to do **in this case**?

Means algorithm **chose bad assignments**

Analogous to $\Gamma \rightarrow \neg p$ **case**

$\Gamma \vdash p$	$\neg\Gamma$ or $\Gamma \rightarrow \neg p$	Can't extend Γ
SAT	**BACKTRACK**	**UNSAT**

DPLL with Theories

0. Put into conjunctive form, simplify
1. If assignment impossible, **backtrack**
2. If statement true, **return assignment**
3. If statement false, **backtrack**
4. Pick an atomic clause, **try assigning True**
5. If it doesn't work, **try assigning False**

Building a Solver

DPLL(T) needs way to test if **assignment is possible**

Assignment: **AND of atomic clauses**

Simpler than testing if **statement** is possible

Splits automated prover into two components

DPLL: Boolean formula

Solver: Test assignment

Proving UNSAT

Assigning tasks to different solvers

Standard Form

$$\neg\ a[x + y := 5][y + x] < 6$$

Standard Form

¬ G **Conjunctive form**

Integer ∧ **G =** **F** < **A**

Array ∧ **F =** **E** [**D**]

Array ∧ **E =** $a[$ **C** $:=$ **B**$]$

Integer ∧ **D =** $y + x$ **One operation**

Integer ∧ **C =** $x + y$

Integer ∧ **B =** 5

Integer ∧ **A =** 6

Standard Form

¬ G **Assignment**

Integer ∧ **G =** **F** < **A**

Array ∧ **F =** **E** [**D**]

Array ∧ **E =** $a[$ **C** $:=$ **B**$]$

Integer ∧ **D =** $y + x$

Integer ∧ **C =** $x + y$

Integer ∧ **B =** 5

Integer ∧ **A =** 6

Standard Form

¬ G **Assignment**

∧ E = $a[$ C $:=$ B$]$

∧ F = E [D]

∧ G = F < A

∧ D = $y + x$

∧ C = $x + y$

∧ B = 5

∧ A = 6

Standard Form

¬ G **Assignment**

∧ E = $a[$ C $:=$ B$]$

∧ F = E [D]

∧ G = F < A

∧ D = $y + x$

∧ C = $x + y$

∧ B = 5

∧ A = 6

Standard Form

$$\neg\ G$$

$$\wedge\quad E = a[\ \ C\ \ := B]$$

$$\wedge\quad F = E\ [\ \ D\ \ ]$$

$$\wedge\quad G = F\ <\ \ A$$

$$\wedge\quad D = y + x$$

$$\wedge\quad C = x + y$$

$$\wedge\quad B = 5$$

$$\wedge\quad A = 6$$

Standard Form

$\neg$ G

$\wedge$ E = $a[$ C $:=$ B$]$

$\wedge$ F = E [D]

$\wedge$ G = F $<$ A

$\wedge$ D = $y + x$

$\wedge$ C = $x + y$

$\wedge$ B = 5

$\wedge$ A = 6

Send to Array solver

If either UNSAT, original impossible

Send to Integer solver

Course Updates

Assignment 2 Recitation

Assignment 2

Expect this to be **harder** than assignment 1

- Start early, **ask questions** on Piazza or after class
- We hope for **fewer installation problems**

Recitation on **Friday 13:00 MEB 3485**

- Will cover invoking Z3, its **input and output format**
- Example of **solving equations** using Z3

Survey Comments

I've noticed **fewer students** filling out the survey

That's why **it's mandatory**!

It helps me correcting misconceptions, improve lectures

Several comments on content being **too theoretical**

It would be helpful to see some examples of code

I'm having a hard time connecting content to how Z3 works

[It] will start to make more sense when we start implementing

Interface Clauses

Ensuring two theories play nice

Standard Form

$\neg$ G

$\wedge$ E = a[C := B]

$\wedge$ F = E [D]

$\wedge$ G = F < A

$\wedge$ D = $y + x$

$\wedge$ C = $x + y$

$\wedge$ B = 5

$\wedge$ A = 6

SAT

a	B	C	D	E	F
?	?	?	?	?	?

How to represent integer values?

SAT

x	y	A	B	C	D	F
0	0	6	5	0	0	7

Abstract Values

Generic data structures: values from another theory

How to represent values of **unknown sort**?

Only one valid operation: **same or different**

Arrays: **abstract keys and abstract values**

$$
\begin{aligned}
?1 &\rightarrow ?2 \\
?3 &\rightarrow ?4 \\
&\vdots
\end{aligned}
$$

Standard Form

¬ G

∧ E = a[C := B]

∧ F = E [D]

∧ G = F < A

∧ D = $y + x$

∧ C = $x + y$

∧ B = 5

∧ A = 6

SAT

a	B	C	D	E	F
?1 → ?2 ?3 → ?4	?5	?1	?3	?1 → ?5 ?3 → ?4	?4

Keys, values represented abstractly

SAT

x	y	A	B	C	D	F
0	0	6	5	0	0	7

Standard Form

¬ G

∧ E = a[C := B]

∧ F = E [D]

∧ G = F < A

∧ D = $y + x$

∧ C = $x + y$

∧ B = 5

∧ A = 6

SAT

a	B	C	D	E	F
?1 → ?2 ?3 → ?4	?5	?1	?3	?1 → ?5 ?3 → ?4	?4

C = D? Theories disagree

SAT

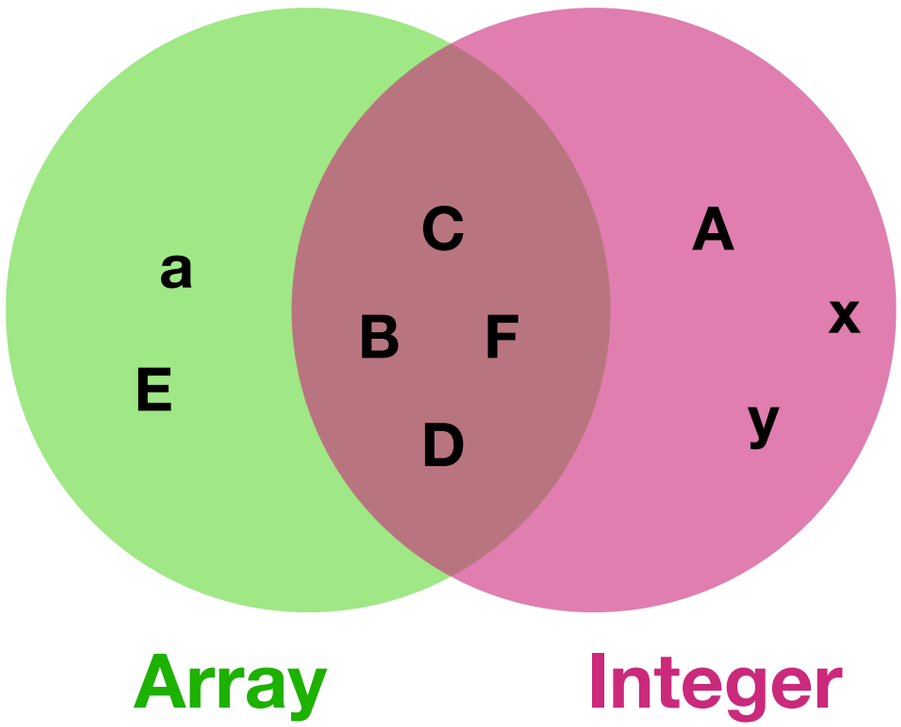
x	y	A	B	C	D	F
0	0	6	5	0	0	3

Interface Clauses

Each sort, operation from **single theory**

Only disagreement possible: **equal or disequal**

Need consistent equality for **shared variables**



Interface Clauses

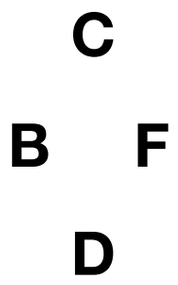
Each sort, operation from **single theory**

Only disagreement possible: **equal or disequal**

Need consistent equality for **shared variables**

Interface Clauses

B = C	**C = D**
B = D	**C = F**
B = F	**D = F**



Consistency

$$B \neq C \vee C \neq D \vee B = D$$
$$B \neq C \vee C \neq F \vee B = F$$
$$\vdots$$

Choose consistent assignment of interface clauses

Truth values added to **both theories**, enforce consistency

Standard Form

¬ G

∧ E = a[C := B]

∧ F = E [D]

∧ G = F < A

∧ D = $y + x$

∧ C = $x + y$

∧ B = 5

∧ A = 6

Choose C = D

SAT

a	B	C	D	E	F
?1 → ?2 ?3 → ?4	?5	?1	?3	?1 → ?5 ?3 → ?4	?4

C = D? Theories disagree

SAT

x	y	A	B	C	D	F
0	0	6	5	0	0	3

Standard Form

$\neg$ G $\wedge$ C = D

$\wedge$ E = a[C := B]

$\wedge$ F = E [D]

$\wedge$ C = D

$\wedge$ G = F < A

$\wedge$ D = $y + x$

$\wedge$ C = $x + y$

$\wedge$ B = 5

$\wedge$ A = 6

$\wedge$ C = D

Choose B = F

SAT

a	B	C	D	E	F
?1 → ?2 ?3 → ?4	?5	?1	?3	?1 → ?5 ?3 → ?4	?5

B = F? Theories disagree

SAT

x	y	A	B	C	D	F
0	0	6	5	0	0	7

Standard Form

$\neg$ G $\wedge$ C = D $\wedge$ B = F

$\wedge$ E = $a[$ C $:=$ B$]$

$\wedge$ F = E [D]

$\wedge$ C = D $\wedge$ B = F

$\wedge$ G = F $<$ A

$\wedge$ D = $y + x$

$\wedge$ C = $x + y$

$\wedge$ B = 5

$\wedge$ A = 6

$\wedge$ C = D $\wedge$ B = F

Ok, try B ≠ F

SAT

a	B	C	D	E	F
?1 → ?2 ?3 → ?4	?5	?1	?1	?1 → ?5 ?3 → ?4	?5

SAT

x	y	A	B	C	D	F
0	0	6	5	0	0	7

Standard Form

$\neg$ G $\wedge$ C = D $\wedge$ B $\neq$ F

$\wedge$ E = $a[$ C $:=$ B$]$

$\wedge$ F = E [D]

$\wedge$ C = D $\wedge$ B $\neq$ F

$\wedge$ G = F $<$ A

$\wedge$ D = $y + x$

$\wedge$ C = $x + y$

$\wedge$ B = 5

$\wedge$ A = 6

$\wedge$ C = D $\wedge$ B $\neq$ F

Ok, try C ≠ D

SAT

a	B	C	D	E	F
?1 → ?2 ?3 → ?4	?5	?1	?1	?1 → ?5 ?3 → ?4	?5

SAT

x	y	A	B	C	D	F
0	0	6	5	0	0	7

Standard Form

$\neg$ G $\wedge$ C $\neq$ D

$\wedge$ E = $a[$ C := B$]$

$\wedge$ F = E [D]

$\wedge$ C $\neq$ D

$\wedge$ G = F $<$ A

$\wedge$ D = $y + x$

$\wedge$ C = $x + y$

$\wedge$ B = 5

$\wedge$ A = 6

$\wedge$ C $\neq$ D

Ok, UNSAT

SAT

a	B	C	D	E	F
?1 → ?2 ?3 → ?4	?5	?1	?1	?1 → ?5 ?3 → ?4	?5

SAT

x	y	A	B	C	D	F
0	0	6	5	0	0	7

Summary of Solvers

Standard Form $\exists x, \exists y, \exists z, p$

Conjunctive Form $(a_1 \vee \neg a_2) \wedge (\neg b_1 \vee \cdots) \wedge \cdots)$

DPLL(T)

Literal Assignment $a_1 \wedge a_2 \wedge \neg b_1 \wedge \cdots$

Nelson-Oppen

$a_1 \wedge \neg b_1 \wedge \cdots$ $a_2 \wedge \cdots$

Solver A Interface Clauses **Solver B**

Next class:

Equality

To do:

- ☐ Course feedback
- ☐ Read Chapter 9
- ☐ Assignment 2

Mixing Theories

Converting first-order problems to a **standard form**

Combining conjunctive form, prefix form, and satisfiability

Separating problems across different theories

Assigning atomic clauses to theories

Interfacing between theories

Matching assumptions in different domains

HASHING

EQUALITY

TO IDENTITY

E-CLASSES

REPRESENTING RECURSIVE REPLACEMENT

REBUILDING

CREATING

CONGRUENCE

CLOSURE

Next class:

Equality

To do:

- ☐ Course feedback
- ☐ Read Chapter 9
- ☐ Assignment 2