#16: How SAT/SMT Solvers Work

Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis

The SAT problem

- **Input:** propositional formula in CNF

literal
= atom or its negation

formula
= conjunction (set) of clauses

$$\boxed{(\neg \boxed{a} \vee \boxed{\neg b}) \wedge (\boxed{b} \vee c) \wedge \boxed{(\neg a \vee \neg c \vee d)} \wedge (b \vee \neg c \vee \neg d) \wedge (a \vee d)}$$

atom
= propositional variable

clause
= disjunction of literals

The SAT problem

- **Problem:** find a *satisfying assignment* (also called a *model*)
  - or determine that the formula is *unsatisfiable*

$$(\neg a \vee \neg b) \wedge (b \vee c) \wedge (\neg a \vee \neg c \vee d) \wedge (b \vee \neg c \vee \neg d) \wedge (a \vee d)$$

a satisfying assignment:

$\{a \mapsto 0, b \mapsto 1, c \mapsto 0, d \mapsto 1\}$

can be written as a set of literals:

$\{\neg a, b, \neg c, d\}$

or as a formula:

$\neg a \wedge b \wedge \neg c \wedge d$

Naive solution

$$(\neg a \vee \neg b) \wedge (b \vee c) \wedge (\neg a \vee \neg c \vee d) \wedge (b \vee \neg c \vee \neg d) \wedge (a \vee d)$$

- Build a truth table!
  - We can't do fundamentally better: it's an NP-complete problem
  - But we can do way better in practice for common instances

$2^{|P|}$

0000	0
0001	0
0010	0
0011	0
0100	0
0101	1
0110	0
0111	1
…	

Intuition: Sudoku

- Easy vs hard: what’s the difference?

7	9					3		
					6	9		
8				3			7	6
			9	6	5			2
		5	4	1	8	7		
4			7	2	3			
6	1			9				8
		2	3					
		9					5	4

		9	7	4	8			
7								
	2		1		9			
		7				2	4	
	6	4		1		5	9	
	9	8				3		
			8		3		2	
								6
			2	7	5	9		

- Most real-world SAT instances allow a lot of inference

DPLL algorithm

[Davis, Putnam '60]
[Davis, Logemann, Loveland '62]

- **State:** current model *M* (a sequence of annotated literals)

decision literal

$$M = \boxed{a^d} \neg b \; c$$

- **Transitions:**
  - **decide** $M \longrightarrow M\, l^d$ **if *l* undefined in *M***
  - **unit-propagate** $M \longrightarrow M\, l$ **if there is a clause where all literals are false except *l*, which is undefined**
  - **backtrack** $M l^d M' \longrightarrow M \neg l$ **if there is a conflicting clause and *M'* has no decision literals**
  - **fail** $M \longrightarrow Unsat$ **if there is a conflicting clause and no decision literals**

DPLL: example

$$\boxed{(\neg a \vee \neg b)} \wedge \boxed{(b \vee c)} \wedge \boxed{(\neg a \vee \neg c \vee d)} \wedge \boxed{(b \vee \neg c \vee \neg d)} \wedge \boxed{(a \vee d)}$$

$M =$	$\emptyset$	decide
	a^d	unit-propagate
	$a^d\ \neg b$	unit-propagate
	$a^d\ \neg b\ c$	unit-propagate
	$a^d\ \neg b\ c\ d$	backtrack
	$\neg a$	unit-propagate
	$\neg a\ d$	decide
	$\neg a\ d \neg c^d$	unit-propagate
	$\neg a\ d \neg c^d\ b$	SAT!

DPLL + clause learning

$$\boxed{(\neg a \vee b)} \wedge \boxed{(\neg c \vee d)} \wedge \boxed{(\neg e \vee \neg f)} \wedge \boxed{(f \vee \neg b \vee \neg e)} \wedge (\neg a \vee \neg e)$$

$M =$ $\emptyset$	decide	
a^d	unit-propagate	
$a^d\ b$	decide	
$a^d\ b\ c^d$	unit-propagate	
$a^d\ b\ c^d\ d$	decide	
$a^d\ b\ c^d\ d\ e^d$	unit-propagate	
$a^d\ b\ c^d\ d\ e^d\ \neg f$	backtrack	Wait, but why?
$a^d\ b\ c^d\ d\ \neg e$		

Bad decision!

DPLL + clause learning

$$(\neg a \vee b) \wedge (\neg c \vee d) \wedge (\neg e \vee \neg f) \wedge (f \vee \neg b \vee \neg e) \wedge (\neg a \vee \neg e)$$

$M =$	$\emptyset$	decide
	a^d	unit-propagate
	$a^d\ b$	decide
	$a^d\ b\ c^d$	unit-propagate
	$a^d\ b\ c^d\ d$	decide
	$a^d\ b\ c^d\ d\ e^d$	unit-propagate
	$a^d\ b\ c^d\ d\ e^d\ \neg f$	**backjump**
	$a^d\ b\ \neg e$	

This lecture

1. Demo: how to use Z3 to
   - solve constraints
   - verify programs
   - synthesize programs
2. How do SAT solvers work?
3. How do SMT solvers work?

Beyond propositional logic

- What if our formula looks like this?

$$(p \land \neg q \lor a = f(b - c)) \land (g(g(b) \neq c \lor a - c \leq 7)$$

  - talks about integers, functions, sets, lists...
- One idea: bit-blast everything and use SAT
  - can only find solutions within bounds
  - very inefficient, so bounds are small
- Better idea: combine SAT with special **solvers** for **theories**
  - they “natively understand” integers, functions, etc.

First-order theories

- theory = <**function symbols**, **predicate symbols**, **axioms**>

ground first-order formulas over
functions and predicates

- **Example:** theory of Equality and Uninterpreted Functions
- EUF = <{**f**, **g**, **h**, ...}, {**=**}, {

$$\forall x. x = x$$
$$\forall x\, y. x = y \;\Rightarrow\; y = x$$
$$\forall x\, y\, z. x = y \wedge y = z \;\Rightarrow\; x = z$$
$$\forall x\, y. x = y \;\Rightarrow\; f(x) = f(y)$$

}>

The SMT problem

literal
= atom or its negation

formula
= conjunction (set) of clauses

theory +

$$\boxed{g(a) = c} \wedge \boxed{\left(\boxed{f(g(a)) \neq f(c)} \vee \boxed{g(a) = d}\right)} \wedge c \neq d$$

atom
= predicate application

clause
= disjunction of literals

Theories for our purpose

- theory = <**function symbols**, **predicate symbols**, ~~axioms~~>

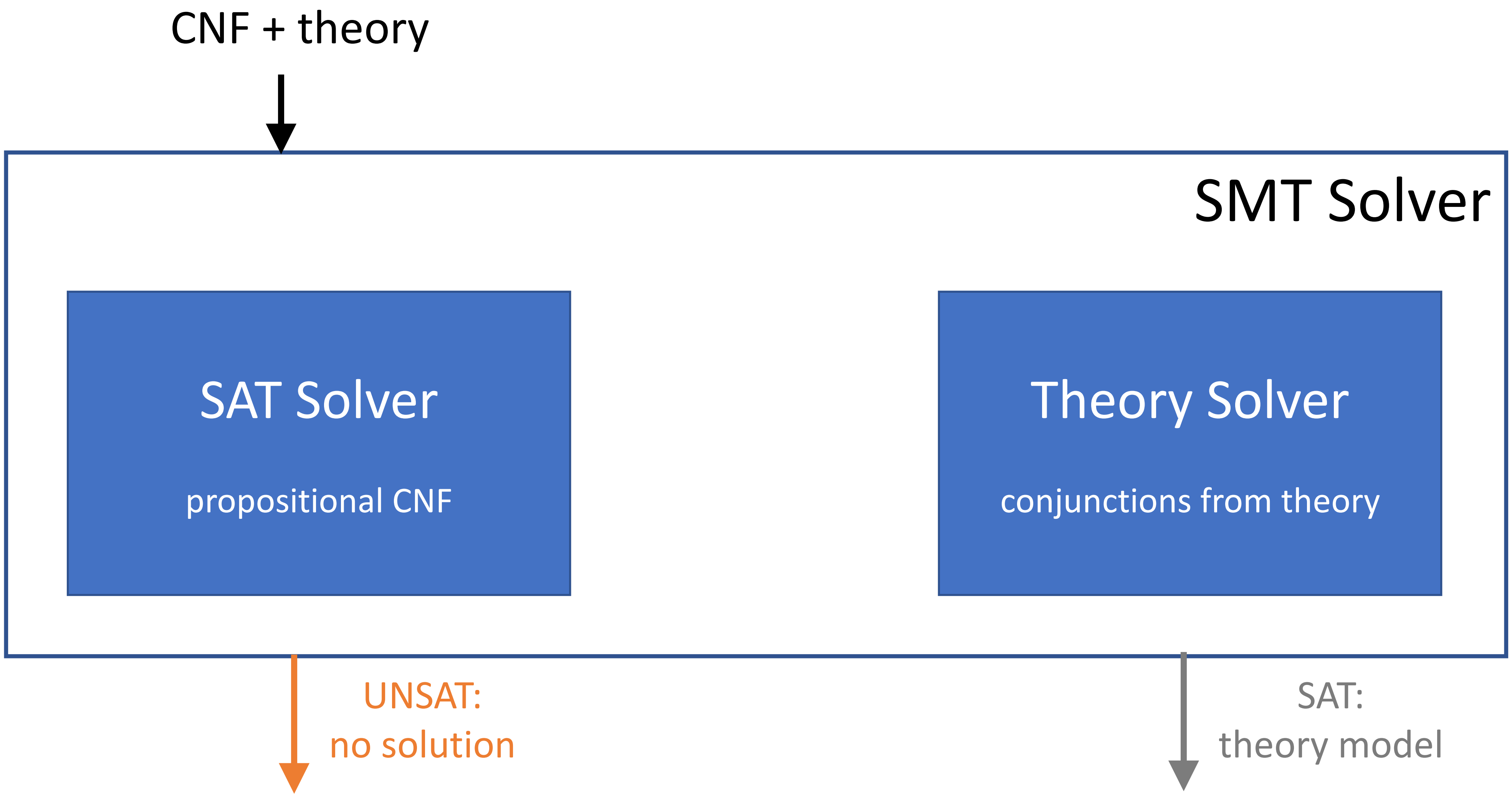
solve
r

can decide consistency of
conjunctions of literals

$$f(a) = c$$
$$f(b) \neq d$$
$$c = d$$
$$a = b$$

$\xrightarrow{\text{EUF solver}}$ Inconsistent!

DPLL(T) architecture



Basic DPLL(T)

$$\boxed{g(a) = c} \wedge (\boxed{f(g(a)) \neq f(c)} \vee \boxed{g(a) = d}) \wedge \boxed{c \neq d}$$

abstract atoms to propositional variables

$$p \wedge (\neg q \vee r) \wedge \neg s$$

$$p \wedge (\neg q \vee r) \wedge \neg s \xrightarrow{\text{SAT solver}} p \; \neg q \; \neg s$$

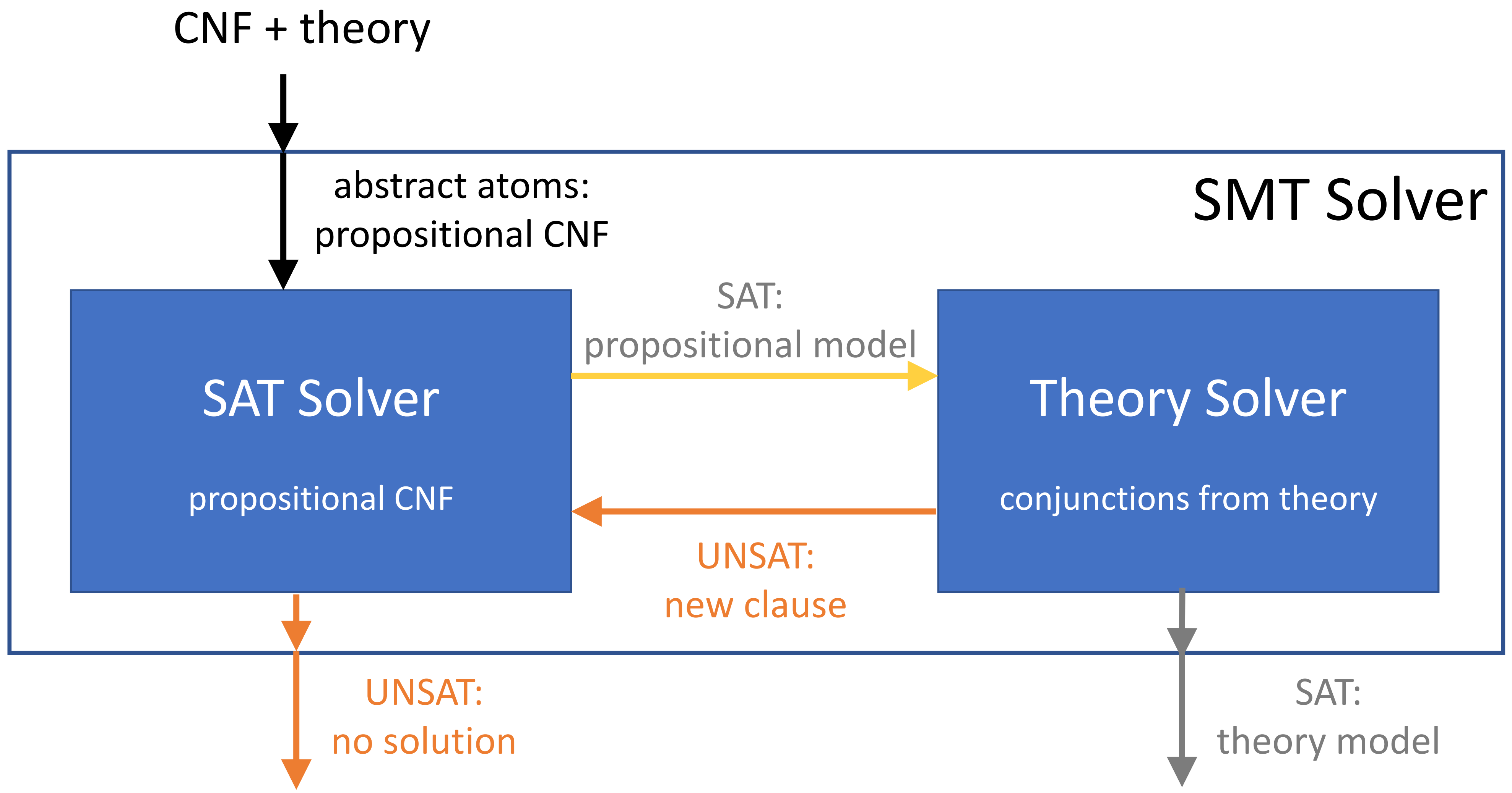
$$\text{Inconsistent!} \xleftarrow{\text{EUF solver}} \boxed{g(a) = c} \quad \boxed{f(g(a)) \neq f(c)} \quad c \neq d$$

$$p \wedge (\neg q \vee r) \wedge \neg s \wedge (\neg p \vee q) \xrightarrow{\text{SAT solver}} p \; q \; r \; \neg s$$

$$\text{Inconsistent!} \xleftarrow{\text{EUF solver}} \boxed{g(a) = c} \quad f(g(a)) = f(c) \quad \boxed{g(a) = d} \quad \boxed{c \neq d}$$

$$p \wedge (\neg q \vee r) \wedge \neg s \wedge (\neg p \vee q) \wedge (\neg p \wedge \neg r \wedge s) \xrightarrow{\text{SAT solver}} \text{Unsat}$$

DPLL(T) architecture



Popular theories

- Equality and Uninterpreted Functions
- EUF = <{**f**, **g**, **h**, ...}, {**=**}, axioms of equality & congruence>
- Linear Integer Arithmetic
- LIA = <{**0**, **1**, ..., +, -}, {**=, ≤**}, axioms of arithmetic>
- Arrays

Arrays = <{**sel**, **store**}, {**=**}, $\forall a\ i\ v.\ \mathrm{sel}(\mathrm{store}(a, i, v), i) = v$

$\forall a\ i\ j\ v. i \neq j \ \Rightarrow\ \mathrm{sel}(\mathrm{store}(a, i, v), j) = \mathrm{sel}(a, j)$ >

Theories can be combined!

Nelson-Oppen combination

Why do we care?

- If we can encode a synthesis problem as SAT/SMT, we can use solvers to do the search for us
- Get some inspiration from how solvers search
  - Unit propagation similar to top-down propagation (pruning through inference of consequences of a guess)
  - Backjumping / clause learning?
    - Feng, Martins, Bastani, Dillig: [Program synthesis using conflict-driven learning](). PLDI'18
  - Coarse-grained reasoning and gradual refinement like in DPLL(T)?
    - Wang, Dillig, Singh: [Program synthesis using abstraction refinement](). POPL'18