Satisfiability Checking for Propositional Logic

Viktor Kuncak, EPFL

`https://lara.epfl.ch/w/fv`

Propositional (Boolean) Logic

Propositional logic is a language for representing Boolean functions $f : \{0,1\}^n \to \{0,1\}$.

- sometimes we write $\bot$ for 0 and $\top$ for 1

Grammar of formulas:

$$P ::= x \mid 0 \mid 1 \mid P \wedge P \mid \neg P \mid P \oplus P \mid P \rightarrow P \mid P \leftrightarrow P$$

where x denotes variables (identifiers). Corresponding Scala trees:

```
sealed abstract class Expr
case class Var(id: Identifier) extends Expr
case class BooleanLiteral(b: Boolean) extends Expr
case class And(e1: Expr, e2: Expr) extends Expr
case class Or(e1: Expr, e2: Expr) extends Expr
case class Not(e: Expr) extends Expr
...
```

Environment and Truth of a Formula

An environment e is a partial map from propositional variables to $\{0,1\}$

For vector of n boolean variables $\bar{p} = (p_1, \ldots, p_n)$ and $\bar{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$, we denote $[\bar{p} \mapsto \bar{v}]$ the environment e given by $e(p_i) = v_i$ for $1 \leq i \leq n$.

We write $e \models F$, and define $[\![F]\!]_e = 1$, to denote that F is true in environment e, otherwise define $[\![F]\!]_e = 0$

Let $e = \{(a,1),(b,1),(c,0)\}$ and F be $a \wedge (\neg b \vee c)$. Then:

$$[\![a \wedge (\neg b \vee c)]\!]_e \;=\; e(a) \wedge (\neg e(b) \vee e(c)) \;=\; 1 \wedge (\neg 1 \vee 0) \;=\; 0$$

The general definition is recursive:

$$\begin{array}{rcl} [\![x]\!]_e &=& e(x) \\ [\![0]\!]_e &=& 0 \\ [\![1]\!]_e &=& 1 \\ [\![F_1 \wedge F_2]\!]_e &=& [\![F_1]\!]_e \wedge [\![F_2]\!]_e \\ [\![\neg F_1]\!]_e &=& \neg [\![F_1]\!]_e \end{array}$$

Note: $\wedge$ and $\neg$ on left and right are different things

Truth of a Formula in Scala

The `interpret` method in `Expr.scala` of Labs 02:

```
def interpret(env: Map[Identifier, Boolean]): Boolean = this match {
  case Var(id) => env(id)
  case BooleanLiteral(b) => b
  case Equal(e1, e2) => e1.interpret(env) == e2.interpret(env)
  case Implies(e1, e2) => !e1.interpret(env) || e2.interpret(env)
  case And(e1, e2) => e1.interpret(env) && e2.interpret(env)
  case Or(e1, e2) => e1.interpret(env) || e2.interpret(env)
  case Xor(e1, e2) => e1.interpret(env) ^ e2.interpret(env)
  case Not(e) => !e.interpret(env)
}
```

Satisfiability Problem

Formula F is *satisfiable*, iff there **exists** e such that $[\![F]\!]_e = 1$.
Otherwise we call F *unsatisfiable*: when there does not exist e such that $[\![F]\!]_e = 1$, that is, for all e, $[\![F]\!]_e = 0$.
Example: let F be $a \wedge (\neg b \vee c)$. Then F is satisfiable, with e.g. $e = \{(a,1),(b,0),(c,0)\}$
Its negation of $\neg F$, is also satisfiable, with e.g. $e = \{(a,0),(b,0),(c,0)\}$

SAT is a problem: given a propositional formula, determine whether it is satisfiable.

The problem is decidable because given F we can compute its variables $FV(F)$ and it suffices to look at the 2^n environments for $n = FV(F)$. The problem is NP-complete, but useful heristics exist.

A SAT solver is a program that, given boolean formula F, either:

- returns **sat**, and, optionally, returns one environment e such that $[\![F]\!]_e = 1$, or
- returns **unsat** and, optionally, returns a **proof** that no satisfying assignment exists

Formal Proof System

We will consider a some set of logical formulas $\mathscr{F}$ (e.g. propositional logic)

Definition

An proof system is $(\mathscr{F}, \text{Infer})$ where $\text{Infer} \subseteq \mathscr{F}^* \times \mathscr{F}$ a decidable set of *inference steps*.

- a set is *decidable* iff there is a program to check if an element belongs to it
- given a set S, notation S^* denotes all finite sequences with elements from S

We schematically write an inference step $((P_1, \ldots, P_n), C) \in \text{Infer}$ by

$$\frac{P_1 \ldots P_n}{C}$$

and we say that from $P_1, \ldots, P_n$ (**premises**) we derive C (**conclusion**).
An inference step is called an *axiom instance* when $n = 0$ (it has no premises).
Given a proof system $(\mathscr{F}, \text{Infer})$, a proof is a finite sequence of inference steps such that, for every inference step, each premise is a conclusion of a previous step.

Proof in a Proof System

Definition

Given $(\mathscr{F}, \text{Infer})$ where $\text{Infer} \subseteq \mathscr{F}^* \times \mathscr{F}$ a **proof** in $(\mathscr{F}, \text{Infer})$ is a finite sequence of inference steps $S_0, \ldots, S_m \in \text{Infer}$ such that, for each S_i where $0 \leq i \leq m$, for each premise P_j of S_i there exists $0 \leq k < i$ such that P_j is the conclusion of S_k.

$$
\begin{array}{lll}
S_0 : & ((), & C_0) \\
& \ldots & \\
S_k : & ((\ldots\ldots\ldots), & \mathbf{P_j}) \\
& \ldots & \\
S_i : & ((\ldots, \mathbf{P_j}, \ldots), & C_i)
\end{array}
$$

Given the definition of the proof, we can replace each premise P_j with the index k where P_j was the conclusion of S_k ($P_j \equiv \text{Conc}(S_k)$)

A proof is then a sequence of elements of $(\{0, 1, \ldots\}^*, \mathscr{F})$ where each S_i is of the form $(k_1, \ldots, k_n, C)$ for $0 \leq k_1, \ldots, k_n < i$ and $(\text{Conc}(S_{k_1}), \ldots, \text{Conc}(S_{k_n}), C) \in \text{Infer}$.

Proofs as Dags

We can view proofs as directed acyclic graphs.

Given a proof as a sequence of steps $(\{0,1,\ldots\}^*, \mathscr{F})$, for each $(k_1,\ldots,k_n,C)$ in the sequence we introduce a node labelled by C, and directed labelled edges $(\text{Conc}(S_{k_j}), j, C)$ for all premises $k_1,\ldots,k_n$.

To check such proof, for each node, follow all of its incoming edges backwards in the order of their indices to find the premises, then check that the inference step is in Infer.

A Minimal Propositional Logic Proof System

Formulas $\mathscr{F}$ defined by $F ::= x \,|\, 0 \,|\, F \rightarrow F$

Shorthand:
$\neg F \equiv F \rightarrow 0$

Inference rules: Infer $= P_2 \cup P_3 \cup \text{MP}$ where: (W: Hilbert system)

$$
\begin{array}{rcll}
P_2 & = & \{((), & F \rightarrow (G \rightarrow F) \quad) \,|\, F, G \in \mathscr{F}\} \\
P_3 & = & \{((), & ((F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H)) \quad) \,|\, F, G, H \in \mathscr{F}\} \\
\text{MP} & = & \{((F \rightarrow G, F), & G \quad) \,|\, F, G \in \mathscr{F}\}
\end{array}
$$

Elements of P_1, P_2, P_3 are all axioms. These are infinite sets, but are given a schematic way and there is an algorithm to check if a given formula satisfies each of the schemas.

Exercise: draw a DAG representing proof of $a \rightarrow a$ where a is a propositional variable.

An Example Proof

Hint: use P_3 for $F \equiv a$, $G \equiv a \rightarrow a$, $H \equiv a$

An Example Proof

Hint: use P_3 for $F \equiv a$, $G \equiv a \longrightarrow a$, $H \equiv a$
Apply MP to the above instance of P_3 and an instance of P_2, then to another instance of P_2.

Derivation is a Proof from Assumptions

Definition

Given $(\mathscr{F}, \text{Infer})$, $\text{Infer} \subseteq \mathscr{F}^* \times \mathscr{F}$ **and a set of assumptions** $A \subseteq \mathscr{F}$, a **derivation from** A in $(\mathscr{F}, \text{Infer})$ is a proof in $(\mathscr{F}, \text{Infer}')$ where:

$$\text{Infer}' = \text{Infer} \cup \{((), F) \mid F \in A\}$$

Thus, assumptions from A are treated just as axioms.

Definition

We say that $F \in \mathscr{F}$ is provable from assumptions A, denoted $A \vdash_{\text{Infer}} F$ iff there exists a derivation from A in Infer that contains an inference step whose conclusion is F.

We write $\vdash_{\text{Infer}} F$ to denote that there exists a proof in Infer containing F as a conslusion (same as $\emptyset \vdash_{\text{Infer}} F$).

Consequence and Soundness in Propositional Logic

Given a set $A \subseteq \mathscr{F}$ where $\mathscr{F}$ are in propositional logic, and $C \in \mathscr{F}$, we say that C is a **semantic consequence** of A, denoted $A \models C$ iff for every environment e that defines all variables in $FV(C) \cup \bigcup_{P \in A} FV(P)$, if $[\![P]\!]_e = 1$ for all $P \in A$, then then $[\![C]\!]_e = 1$.

Definition

Given $(\mathscr{F}, \text{Infer})$ where $\mathscr{F}$ are propositional, step $((P_1 \ldots P_n), C) \in \text{Infer}$ is **sound** iff $\{P_1, \ldots, P_n\} \models C$. Proof system Infer is sound if every inference step is sound.

For axioms, this definition reduces to saying that C is true for all interpretations, i.e., that C is a valid formula (tautology).

Theorem

Let $(\mathscr{F}, Infer)$ where $\mathscr{F}$ are propositional logic formulas. If every inference rule in Infer is sound, then $A \vdash_{Infer} F$ implies $A \models F$.

Proof is immediate by induction on the length of the formal proof.
Consequence: $\vdash_{\text{Infer}} F$ implies F is a tautology.

A Proof System with Decision and Simplification

Propositional formulas F and G are semantically equivalent if $F \models G$ and $G \models F$.

Case analysis proof rule $((F,G), F[x:=0] \vee G[x:=1]) \mid F, G \in \mathscr{F}, x-\text{variable}\}$:

$$\frac{F \qquad\qquad G}{F[x:=0] \vee G[x:=1]}$$

Proof of soundness: consider an environment e (that defines x as well as $FV(F) \cup FV(G)$), and assume $[\![F]\!]_e = 1$ and $[\![G]\!]_e = 1$.

- If $e(x) = 0$, then $[\![F[x:=0]]\!]_e = [\![F]\!]_e = 1$.
- If $e(x) = 1$, then $[\![G[x:=1]]\!]_e = [\![G]\!]_e = 1$.

Simplification rules that preserve equivalence can be applied: $0 \wedge F \rightsquigarrow 0$, $1 \wedge F \rightsquigarrow F$, $0 \vee F \rightsquigarrow F$, $1 \vee F \rightsquigarrow 1$, $\neg 0 \rightsquigarrow 1$, $\neg 1 \rightsquigarrow 0$.
Introduce inferences $\{((F), F') \mid F'$ is simplified $F\}$. These rules are also sound. Call this Infer_D.

Example Derivation

Derivation from $A = \{a \wedge b,\ \neg b \vee \neg a\}$. Draw the arrows to get a proof DAG

$a \wedge b$

$\neg b \vee \neg a$

$(0 \wedge b) \vee (1 \wedge b)$

$(a \wedge 0) \vee (a \wedge 1)$

b

a

$0 \vee (\neg 1 \vee \neg a)$

$\neg a$

0

Example Derivation

Derivation from $A = \{a \wedge b,\ \neg b \vee \neg a\}$. Draw the arrows to get a proof DAG

$$a \wedge b$$

$$\neg b \vee \neg a$$

$$(0 \wedge b) \vee (1 \wedge b)$$

$$(a \wedge 0) \vee (a \wedge 1)$$

$$b$$

$$a$$

$$0 \vee (\neg 1 \vee \neg a)$$

$$\neg a$$

$$0$$

This derivation shows that: $A \vdash 0$

Proving Unsatisfiability

A *set* A of formulas is *satisfiable* if there exists e such that, for every $F \in A$, $[\![F]\!]_e = 1$.

- when $A = \{F_1, \ldots, F_n\}$ the notion is the same as the satisfiability of $F_1 \wedge \ldots \wedge F_n$

Otherwise, we call the set A *unsatisfiable*.

Theorem (Refutation Soundness)

If $A \vdash_{Infer_D} 0$ then A is unsatisfiable.

Follows from soundness of Infer_D

More interestingly:

Theorem (Refutation Completeness)

If a finite set A is unsatisfiable, then $A \vdash_{Infer_D} 0$

Proof hint: take conjunction of formulas in A and existentially quantify it to get A'. What is the relationship of the truth of A' and the satisfiability of A? For a conjunction of formulas F, can you express $\exists x.F$ using Infer_D?

Conjunctive Form, Literals, and Clauses

A propositional *literal* is either a variable (x) or its negation ($\neg x$).
A *clause* is a disjunction of literals.
For convenience, we can represent clause as a finite *set of literals* (because of associativity, commutativity, and idempotence of $\vee$).

Example: $a \vee \neg b \vee c$ represented as $\{a, \neg b, c\}$

If C is a clause then $[\![C]\!]_e = 1$ iff there exists a literal $l \in C$ such that $[\![l]\!]_e = 1$. We represent 0 using the empty clause $\emptyset$.
As for any formulas, a finite set of clauses A can be interpreted as a conjunction.
Thus, a set of clauses can be viewed as a formula in conjunctive normal form:

$$A = \{\{a\},\ \{b\},\ \{\neg a, \neg b\}\}$$

represents the formula

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

Resolution on Clauses as a Proof System

$a \lor b \lor c$ $\quad$ $d \lor \neg c$	$\{a, b, c\}$ $\quad$ $\{d, \neg c\}$
$(a \lor b \lor 0) \lor (d \lor \neg 1))$	
$a \lor b \lor d$	$\{a, b, d\}$

Clausal resolution rule (transitivity of implication, or decision rule for clauses):

$$\frac{C_1 \cup \{x\} \quad C_2 \cup \{\neg x\}}{C_1 \cup C_2}$$

resolve two clauses with respect to x

Theorem (Soundness)

Clausal resolution is sound for all clauses C_1, C_2 and propositional variable x,
$\{C_1 \cup \{x\}, C_2 \cup \{\neg x\}\} \models C_1 \cup C_2$.

Theorem (Refutational Completeness)

A finite set of clauses A is satisfiable if and only if there exists a derivation of the empty clause from A using clausal resolution.

Exercise

Use resolution to prove that the following formula is valid:

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

Exercise

Use resolution to prove that the following formula is valid:

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

Prove that its negation is unsatisfiable set of clauses:

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

Exercise

Use resolution to prove that the following formula is valid:

$$\neg(a \land b \land (\neg a \lor \neg b))$$

Prove that its negation is unsatisfiable set of clauses:

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

$\{\neg b\}$

Exercise

Use resolution to prove that the following formula is valid:

$$\neg(a \land b \land (\neg a \lor \neg b))$$

Prove that its negation is unsatisfiable set of clauses:

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

$\{\neg b\}$

$\emptyset$

Unit Resolution

A *unit clause* is a clause that has precisely one literal; it's of the form $\{L\}$
Given a literal L, its dual $\overline{L}$ is defined by $\overline{x} = \neg x$, $\overline{\neg x} = x$.

Unit resolution is a special case of resolution where at least one of the clauses is a unit clause:

$$\frac{C \qquad \{L\}}{C \setminus \{\overline{L}\}}$$

Soundness: if L is true, then $\overline{L}$ is false, so it can be deleted from a disjunction C.

Subsumption: when applying resolution, if we obtain a clause $C' \subseteq C$ that is subset of a previosly derived one, we can delete C so we do not consider it any more. Any use of C can be replaced by use of C' with progress towards $\emptyset$ at least as good.

Unit resolution with $\{L\}$ can remove all occurences of L and $\overline{L}$ from our set.

Constructing a Conjunctive Normal Form

How would be transform this formula into a set of clauses:

$$\neg(((c \wedge a) \vee (\neg c \wedge b)) \longleftrightarrow ((c \rightarrow b) \wedge (\neg c \rightarrow b)))$$

Which equivalences are guaranteed to produce a conjunctive normal form?

$$\begin{array}{rcl} \neg(F_1 \wedge F_2) & \longleftrightarrow & (\neg F_1)\neg(\neg F_2) \\ F_1 \wedge (F_2 \vee F_3) & \longleftrightarrow & (F_1 \wedge F_2) \vee (F_1 \vee F_3) \\ F_1 \vee (F_2 \wedge F_3) & \longleftrightarrow & (F_1 \vee F_2) \wedge (F_1 \vee F_3) \end{array}$$

Constructing a Conjunctive Normal Form

How would be transform this formula into a set of clauses:

$$\neg(((c \wedge a) \vee (\neg c \wedge b)) \longleftrightarrow ((c \rightarrow b) \wedge (\neg c \rightarrow b)))$$

Which equivalences are guaranteed to produce a conjunctive normal form?

$$\begin{array}{rcl}
\neg(F_1 \wedge F_2) & \longleftrightarrow & (\neg F_1)\neg(\neg F_2) \\
F_1 \wedge (F_2 \vee F_3) & \longleftrightarrow & (F_1 \wedge F_2) \vee (F_1 \vee F_3) \\
F_1 \vee (F_2 \wedge F_3) & \longleftrightarrow & (F_1 \vee F_2) \wedge (F_1 \vee F_3)
\end{array}$$

What is the complexity of such transformation in the general case?

Constructing a Conjunctive Normal Form

How would be transform this formula into a set of clauses:

$$\neg(((c \land a) \lor (\neg c \land b)) \longleftrightarrow ((c \rightarrow b) \land (\neg c \rightarrow b)))$$

Which equivalences are guaranteed to produce a conjunctive normal form?

$$\begin{array}{rcl} \neg(F_1 \land F_2) & \longleftrightarrow & (\neg F_1)\neg(\neg F_2) \\ F_1 \land (F_2 \lor F_3) & \longleftrightarrow & (F_1 \land F_2) \lor (F_1 \lor F_3) \\ F_1 \lor (F_2 \land F_3) & \longleftrightarrow & (F_1 \lor F_2) \land (F_1 \lor F_3) \end{array}$$

What is the complexity of such transformation in the general case?
Are there efficient algorithms for checking satisfiability of formulas in *disjunctive* normal form (disjunctions of conjunctions of literals)?
When checking satisfiability, is conversion into *conjunctive* normal form any better than disjunctive normal form?

Equivalence and Equisatisfiability

Formulas F_1 and F_2 are **equivalent** iff: $F_1 \models F_2$ and $F_2 \models F_1$

Formulas F_1 and F_2 are **equisatisfiable** iff: F_1 is satisfiable whenever F_2 is satisfiable.

Equivalent formulas are always equisatisfiable, but converse is not the case in general. For example, formulas a and b are equisatisfiable, because they are both satisfiable.

Consider these two formulas:

- F_1: $(a \land b) \lor c$
- F_2: $(x \longleftrightarrow (a \land b)) \land (x \lor c)$

They are equisatisfiable but not equivalent. For example, given $e = \{(a,1),(b,1),(c,0),(x,0)\}$, $[\![F_1]\!]_e = 1$ whereas $[\![F_2]\!]_e = 0$. Interestingly, every choice of a, b, c that makes F_1 true can be extended to make F_2 true appropriately, if we choose x as $[\![a \land b]\!]_e$.

Flatenning as Satisfiability Preserving Transformation

Observation: Let F be a formula, G another formula, and $x \notin FV(F)$ a propositional variable. Let $F[G := x]$ denote the result of replacing an occurence of formula G inside F with x. Then F is equisatisfiable with

$$(x = G) \wedge F[G := x]$$

(Here, $=$ denotes $\longleftrightarrow$.)

Proof of equisatisfiability: a satisfying assignment for new formula is also a satisfying assignment for the old one. Conversely, since x does not occur in F, if $[\![F]\!]_e = 1$, we can change $e(x)$ to be defined as $[\![G]\!]_e$, which will make the new formula true.

(A transformation that produces an equivalent formula: *equivalence preserving*.)
A transformation that produces an equisatisfiable formula: *satisfiability preserving*.
Flattening is this satisfiability preserving transformation in any formalism that supports equality (here: equivalence): pick a subformula and given it a name by a fresh variable, applying the above observation.
Strategy: apply transformation from smallest non-variable subformulas.

Tseytin's Transformation (see also Calculus of Computation, Section 1.7.3)

Consider formula with $\neg, \wedge, \vee, \rightarrow, =, \oplus$

- Push negation into the propositional variables using De Morgan's laws and switching between $\oplus$ and $=$.
- Repeat: flatten an occurrence of a binary connective whose arguments are literals
- In the resulting conjunction, express each equivalence as a conjunction of clauses:

conjunct	clauses
$x = (a \wedge b)$	$\{\neg x, a\}, \{\neg x, b\}, \{\neg a, \neg b, x\}$
$x = (a \vee b)$	$\{\neg x, a, b\}, \{\neg a, x\}, \{\neg b, x\}$
$x = (a \rightarrow b)$	
$x = (a = b)$	
$x = (a \oplus b)$	

Exercise: Complete the missing entries. Are the rules in the last step equivalence preserving or only equisatisfiability preserving? Why is the resulting algorithm polynomial?

Example: Find an Equisatisfiable CNF

$$\neg(((c \land a) \lor (\neg c \land b)) \leftrightarrow ((c \to b) \land (\neg c \to b)))$$

SAT Solvers

A SAT solver takes as input a set of clauses.

To check satisfiability, convert to equisatisfiable set of clauses in polynomial time using Tseytin's transformation.

To check validity of a formula, take negation, check satisfiability, then negate the answer.

How should we check satisfiability of a set of clauses?

- resolution on clauses, favoring unit resolution and applying subsumption (complete)
  Davis and Putnam, 1960
- truth table method: pick one value, then other (fast and space efficient)

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm Sketch

```
def DPLL(S: Set[Clause]) : Bool =
  val S' = subsumption(UnitProp(S))
  if ∅ ∈ S' then false
  else if S' has only unit clauses then true
  else
    val L = a literal from a clause of S' where L ∉ S'
    DPLL(S' ∪ L) || DPLL(S' ∪ complement(L))

def UnitProp(S: Set[Clause]): Set[Clause] = // Unit Propagation (BCP)
  if C ∈ S, unit U ∈S, resolve(U,C) ∉ S
  then UnitProp((S - C) ∪ {resolve(U,C)}) else S

def subsumption(S: Set[Clause]): Set[Clause] =
  if C1,C2 ∈ S such that C1 ⊆ C2
  then subsumption(S - {C2}) else S
```

First Approach: Resolution

a ∨ ¬b ∨ f

¬a ∨ ¬c ∨ d ∨ ¬e

First Approach: Resolution

$a \lor \neg b \lor f$

$\neg a \lor \neg c \lor d \lor \neg e$

First Approach: Resolution

$a \vee \neg b \vee f$ $\quad\quad$ $\neg a \vee \neg c \vee d \vee \neg e$

$\neg b \vee f \vee \neg c \vee d \vee \neg e$

First Approach: Resolution

$a \lor \neg b \lor f$ $\qquad$ $\neg a \lor \neg c \lor d \lor \neg e$

$$\neg b \lor f \lor \neg c \lor d \lor \neg e$$

- Resolution eliminates one variable by producing a new clause (*resolvent*) from complementary ones.

Resolution

$(a \lor b) \land (a \lor \neg b) \land (\neg a \lor c) \land (\neg a \lor \neg c)$

Resolution

$(a \lor b) \land (a \lor \neg b) \land (\neg a \lor c) \land (\neg a \lor \neg c)$

Resolution

$$(a \lor b) \land (a \lor \neg b) \land (\neg a \lor c) \land (\neg a \lor \neg c)$$

$$a \land (\neg a \lor c) \land (\neg a \lor \neg c)$$

Resolution

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

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

Resolution

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

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

$$c \wedge \neg c$$

Resolution

$$(a \lor b) \land (a \lor \neg b) \land (\neg a \lor c) \land (\neg a \lor \neg c)$$

$$a \land (\neg a \lor c) \land (\neg a \lor \neg c)$$

$$c \land \neg c$$

(Part of) Davis Putnam Algorithm

- (Also: when a variable appears in only one polarity, remove all clauses containing it.)

- M. Davis, H. Putnam, *A computing procedure for quantification theory*, JACM, 1960.

- Problem: space explosion!

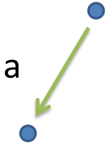
- DP is *proof-oriented*. Current algorithms are *model-oriented*.

Backtracking Search

$$
\begin{aligned}
&(\phantom{\neg a \lor{}} b \lor \neg c) \\
\land\ &(\neg a \lor b \lor c) \\
\land\ &(\neg a \lor \neg b)
\end{aligned}
$$

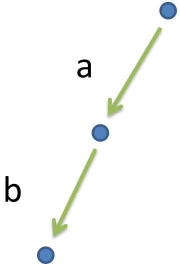
Backtracking Search

$$
\begin{aligned}
& (\quad\quad\quad\;\; b \lor \neg c) \\
\land & (\neg a \lor \; b \lor \;\; c) \\
\land & (\neg a \lor \neg b \quad\quad\;\;)
\end{aligned}
$$



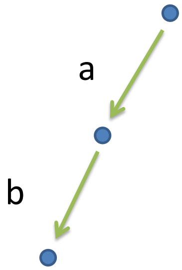
Backtracking Search

$$
\begin{aligned}
& (b \lor \neg c) \\
\land\ & (\neg a \lor b \lor c) \\
\land\ & (\neg a \lor \neg b)
\end{aligned}
$$



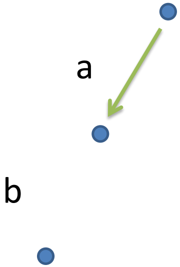
Backtracking Search

$$
\begin{aligned}
&(\qquad\quad b \lor \neg c) \\
\land\ &(\neg a \lor b \lor c) \\
\land\ &(\neg a \lor \neg b \qquad) \; !
\end{aligned}
$$



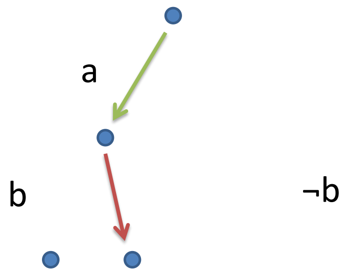
Backtracking Search

$$(b \lor \neg c)$$
$$\land\ (\neg a \lor b \lor c)$$
$$\land\ (\neg a \lor \neg b)$$



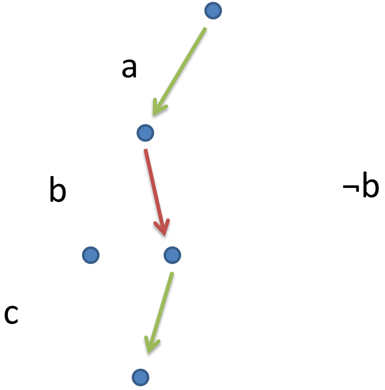
Backtracking Search

$$(\; b \lor \neg c\;) \land (\neg a \lor b \lor c) \land (\neg a \lor \neg b)$$



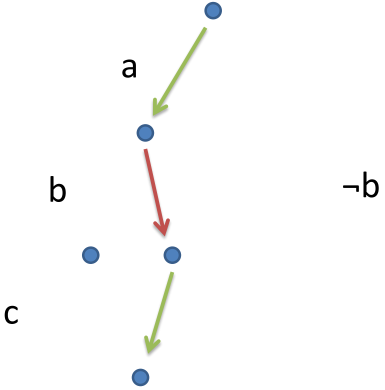
Backtracking Search

$$
\begin{aligned}
& (\; b \lor \neg c\;) \\
\land\; & (\;\neg a \lor b \lor c\;) \\
\land\; & (\;\neg a \lor \neg b\;)
\end{aligned}
$$



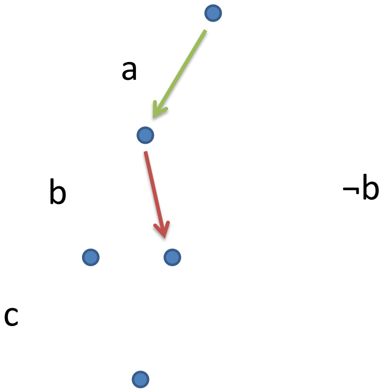
Backtracking Search

$$(\quad b \lor \neg c) \; !$$
$$\land (\neg a \lor b \lor c)$$
$$\land (\neg a \lor \neg b \quad)$$



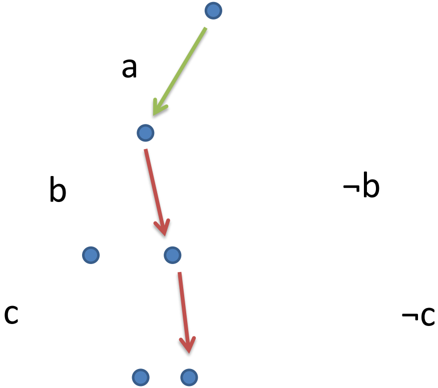
Backtracking Search

$$
\begin{aligned}
& (\phantom{\neg a \vee{}} b \vee \neg c) \\
\wedge\ & (\neg a \vee b \vee c) \\
\wedge\ & (\neg a \vee \neg b)
\end{aligned}
$$



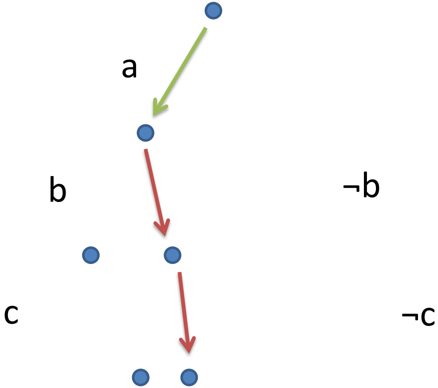
Backtracking Search

$$
\begin{aligned}
&(\quad\quad\quad b \lor \neg c\,) \\
\land\, &(\neg a \lor b \lor c\,) \\
\land\, &(\neg a \lor \neg b\quad\quad\,)
\end{aligned}
$$



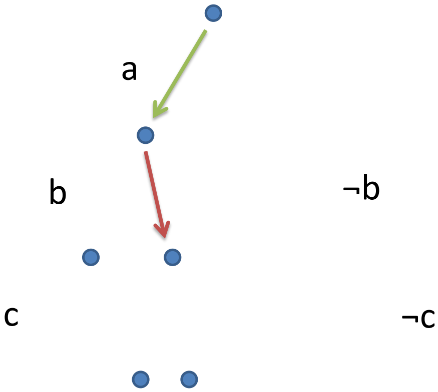
Backtracking Search

$(\quad b \lor \neg c)$
$\land (\neg a \lor b \lor c)$!
$\land (\neg a \lor \neg b \quad)$



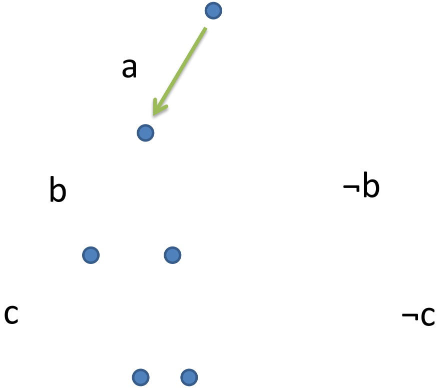
Backtracking Search

$$
\begin{aligned}
& (\phantom{\neg a \lor{}} b \lor \neg c) \\
\land\ & (\neg a \lor b \lor c) \\
\land\ & (\neg a \lor \neg b)
\end{aligned}
$$



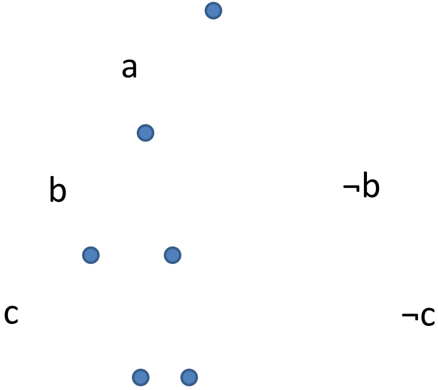
Backtracking Search

$$(b \lor \neg c)$$
$$\land (\neg a \lor b \lor c)$$
$$\land (\neg a \lor \neg b)$$



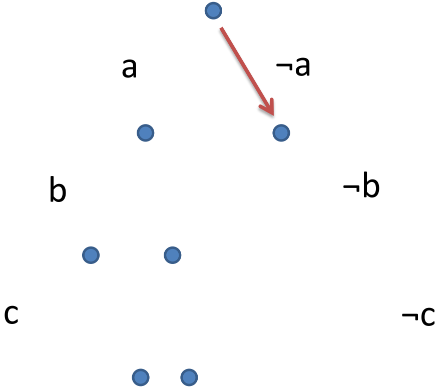
Backtracking Search

$$
\begin{aligned}
& (b \lor \neg c) \\
\land\ & (\neg a \lor b \lor c) \\
\land\ & (\neg a \lor \neg b)
\end{aligned}
$$



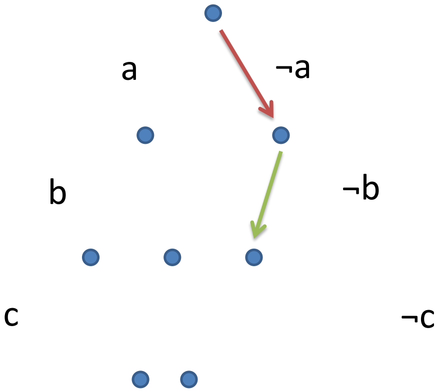
Backtracking Search

$$(\quad\quad b \lor \neg c\,)$$
$$\land\,(\,\neg a \lor b \lor c\,)$$
$$\land\,(\,\neg a \lor \neg b\quad\,)$$



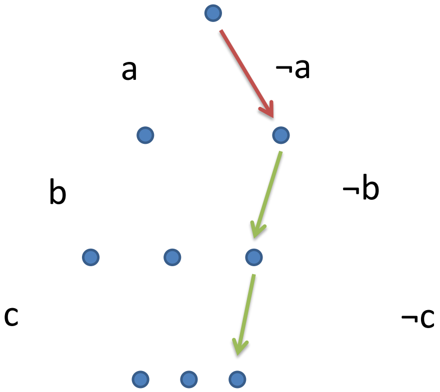
Backtracking Search

$$(\qquad\qquad b \lor \neg c)$$
$$\land (\neg a \lor b \lor c)$$
$$\land (\neg a \lor \neg b \qquad)$$



Backtracking Search

$$
\begin{aligned}
&(\ b \lor \neg c\ ) \\
\land\ &(\ \neg a \lor b \lor c\ ) \\
\land\ &(\ \neg a \lor \neg b\ )
\end{aligned}
$$



Boolean Constraint Propagation

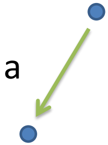
- *"When all but one literal are falsified, it becomes implied."*

$$
\begin{aligned}
& (\qquad\quad b \lor \neg c\ ) \\
\land\ & (\ \neg a \lor \ b \lor \ c\ ) \\
\land\ & (\ \neg a \lor \neg b \qquad)
\end{aligned}
$$

Boolean Constraint Propagation

- *"When all but one literal are falsified, it becomes implied."*

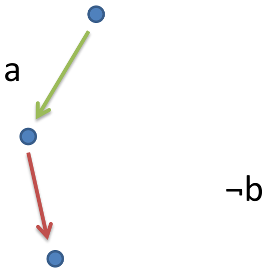
$$
\begin{aligned}
&(\quad\quad\quad\;\; b \lor \neg c\,) \\
\land\, &(\neg a \lor b \lor \;\; c\,) \\
\land\, &(\neg a \lor \neg b \quad\quad\;\;)
\end{aligned}
$$



Boolean Constraint Propagation

- *"When all but one literal are falsified, it becomes implied."*

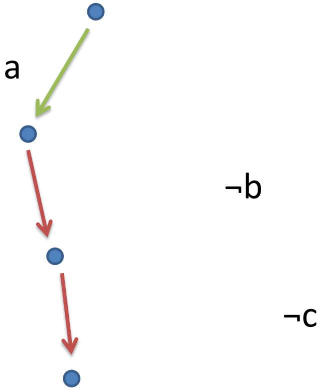
$$
\begin{aligned}
&(\quad\quad\quad\ \ b \lor \neg c\,) \\
\land\ &(\,\neg a \lor\ b \lor\ \ c\,) \\
\land\ &(\,\neg a \lor \neg b\quad\quad\ \ )
\end{aligned}
$$



Boolean Constraint Propagation

- *"When all but one literal are falsified, it becomes implied."*

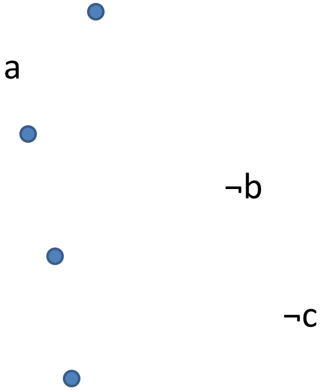
$$
\begin{aligned}
&(b \lor \neg c) \\
\land &(\neg a \lor b \lor c) \\
\land &(\neg a \lor \neg b)
\end{aligned}
$$



Boolean Constraint Propagation

- *"When all but one literal are falsified, it becomes implied."*

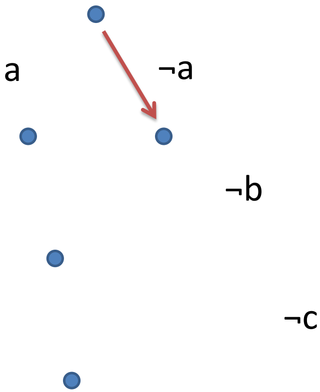
$$
\begin{aligned}
& (\ b \lor \neg c\ ) \\
\land\ & (\ \neg a \lor b \lor c\ ) \\
\land\ & (\ \neg a \lor \neg b\ )
\end{aligned}
$$



Boolean Constraint Propagation

- *"When all but one literal are falsified, it becomes implied."*

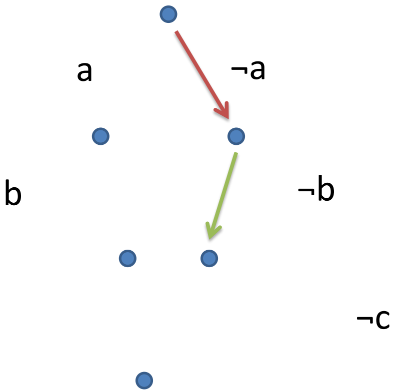
$$
\begin{aligned}
&(\qquad\quad\; b \lor \neg c\,) \\
\land\, &(\neg a \lor b \lor c\,) \\
\land\, &(\neg a \lor \neg b \qquad\;)
\end{aligned}
$$



Boolean Constraint Propagation

- *"When all but one literal are falsified, it becomes implied."*

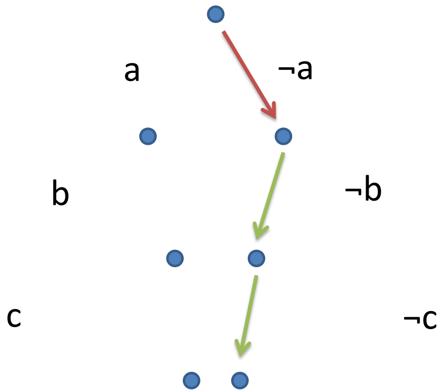
$$
\begin{aligned}
&(\quad b \lor \neg c) \\
\land\ &(\neg a \lor b \lor c) \\
\land\ &(\neg a \lor \neg b\quad)
\end{aligned}
$$



Boolean Constraint Propagation

- *“When all but one literal are falsified, it becomes implied.”*

$$
\begin{aligned}
& (\quad\quad\quad b \lor \neg c\ ) \\
\land & (\neg a \lor b \lor c\ ) \\
\land & (\neg a \lor \neg b \quad\quad\ )
\end{aligned}
$$



Two-watched-literal Scheme for BCP

- BCP can cut the search tree dramatically…
- …but checking each clause for potential implications is expensive.

- Observation: as long as at least two literals in a clause are “not false”, that clause does not imply any new literal.
- Idea: for each clause, try to maintain that invariant.

Cutting Deeper: Learning

- Idea: compute new clauses that are logically implied, and that may trigger more BCP.

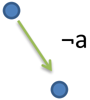
- Use an *implication graph*. When a conflict is derived, look for a *small explanation*.

Learning

$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

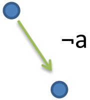
Learning

$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$



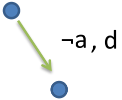
Learning

(a ∨ d)
∧ (a ∨ ¬c ∨ ¬h)
∧ (a ∨ h ∨ ¬m)
∧ (b ∨ k)
∧ (¬g ∨ ¬c ∨ i)
∧ (¬g ∨ h ∨ ¬i)
∧ (g ∨ h ∨ ¬j)
∧ (g ∨ j ∨ ¬m)



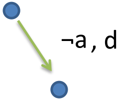
Learning

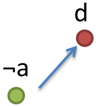
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$



Learning

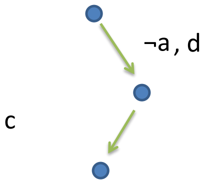
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

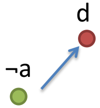




Learning

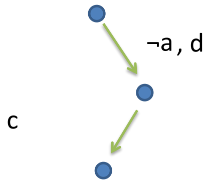
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

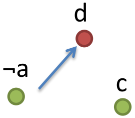




Learning

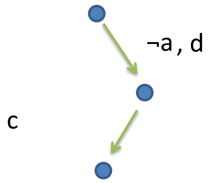
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

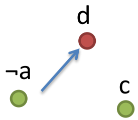




Learning

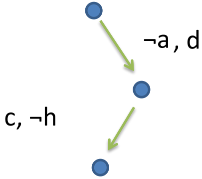
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

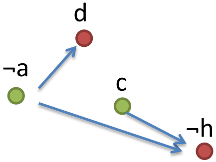




Learning

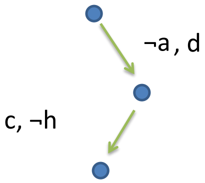
$$(a \lor d)$$
$$\land\ (a \lor \neg c \lor \neg h)$$
$$\land\ (a \lor h \lor \neg m)$$
$$\land\ (b \lor k)$$
$$\land\ (\neg g \lor \neg c \lor i)$$
$$\land\ (\neg g \lor h \lor \neg i)$$
$$\land\ (g \lor h \lor \neg j)$$
$$\land\ (g \lor j \lor \neg m)$$

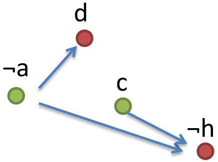




Learning

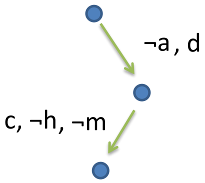
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

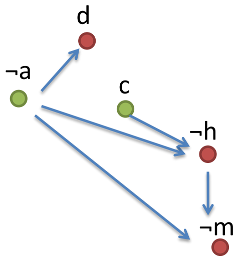




Learning

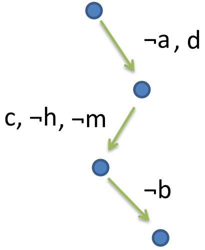
(a ∨ d)
∧ (a ∨ ¬c ∨ ¬h)
∧ (a ∨ h ∨ ¬m)
∧ (b ∨ k)
∧ (¬g ∨ ¬c ∨ i)
∧ (¬g ∨ h ∨ ¬i)
∧ (g ∨ h ∨ ¬j)
∧ (g ∨ j ∨ ¬m)

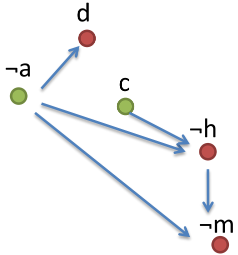




Learning

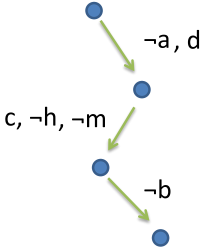
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

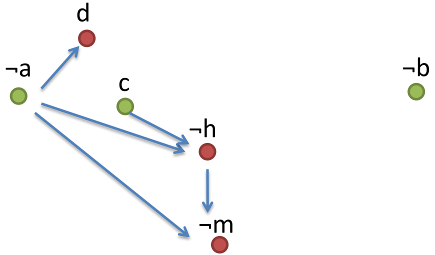




Learning

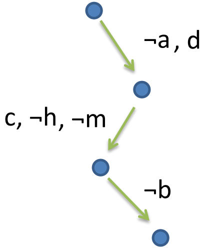
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

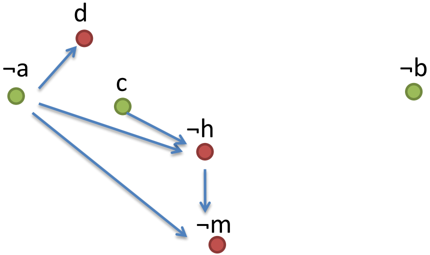




Learning

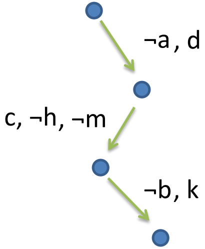
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

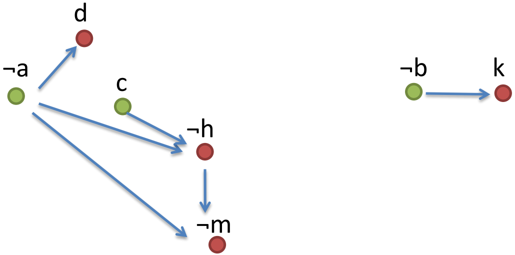




Learning

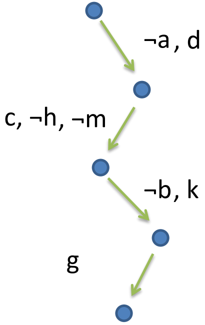
(a ∨ d)
∧ (a ∨ ¬c ∨ ¬h)
∧ (a ∨ h ∨ ¬m)
∧ (b ∨ k)
∧ (¬g ∨ ¬c ∨ i)
∧ (¬g ∨ h ∨ ¬i)
∧ (g ∨ h ∨ ¬j)
∧ (g ∨ j ∨ ¬m)

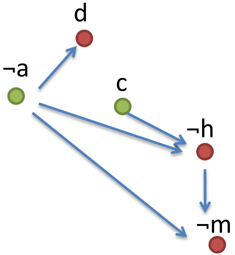




Learning

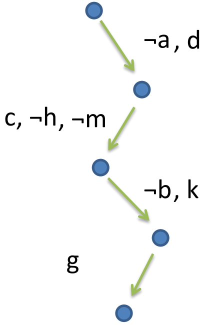
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

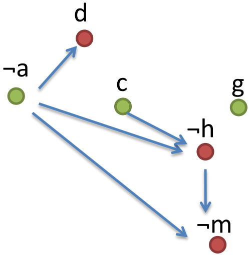




Learning

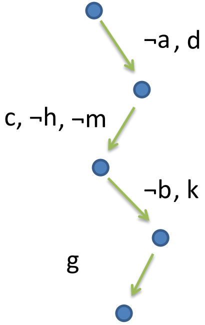
$(a \lor d)$
$\land\ (a \lor \neg c \lor \neg h)$
$\land\ (a \lor h \lor \neg m)$
$\land\ (b \lor k)$
$\land\ (\neg g \lor \neg c \lor i)$
$\land\ (\neg g \lor h \lor \neg i)$
$\land\ (g \lor h \lor \neg j)$
$\land\ (g \lor j \lor \neg m)$

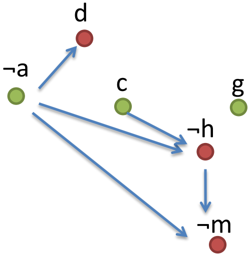




Learning

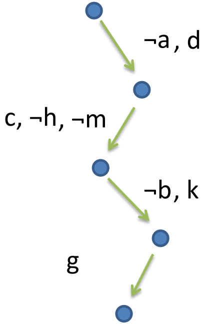
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

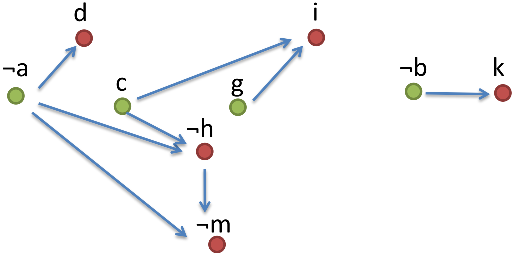




Learning

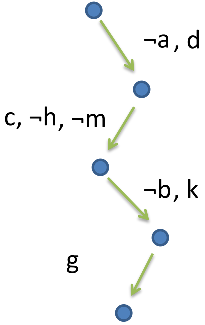
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

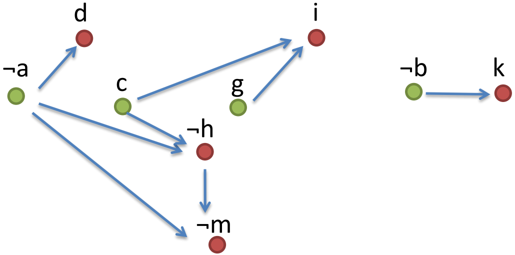




Learning

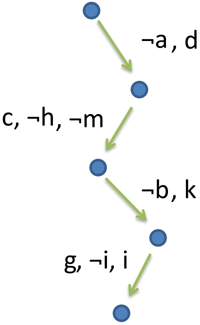
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

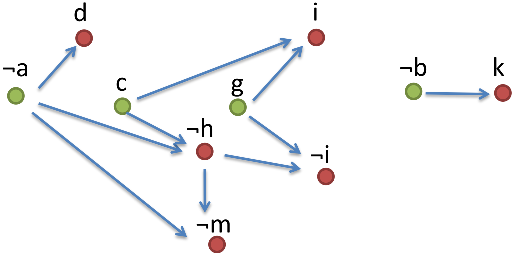




Learning

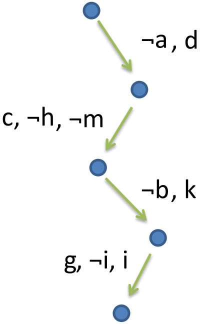
$(a \lor d)$
$\land\ (a \lor \neg c \lor \neg h)$
$\land\ (a \lor h \lor \neg m)$
$\land\ (b \lor k)$
$\land\ (\neg g \lor \neg c \lor i)$
$\land\ (\neg g \lor h \lor \neg i)$
$\land\ (g \lor h \lor \neg j)$
$\land\ (g \lor j \lor \neg m)$

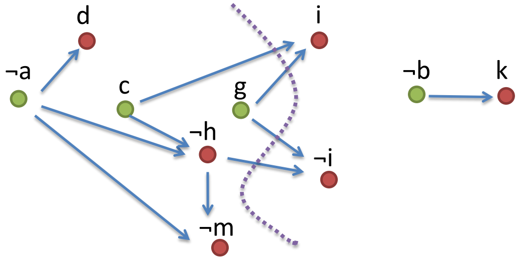




Learning

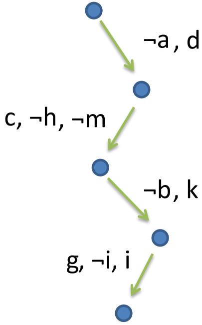
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$

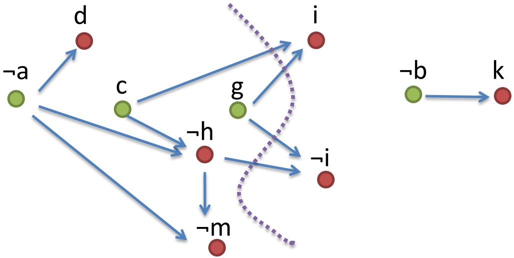




Learning

$(a \lor d)$
$\land\ (a \lor \neg c \lor \neg h)$
$\land\ (a \lor h \lor \neg m)$
$\land\ (b \lor k)$
$\land\ (\neg g \lor \neg c \lor i)$
$\land\ (\neg g \lor h \lor \neg i)$
$\land\ (g \lor h \lor \neg j)$
$\land\ (g \lor j \lor \neg m)$

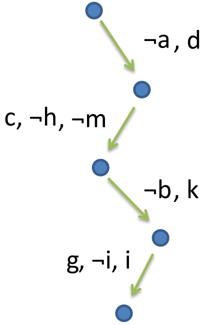


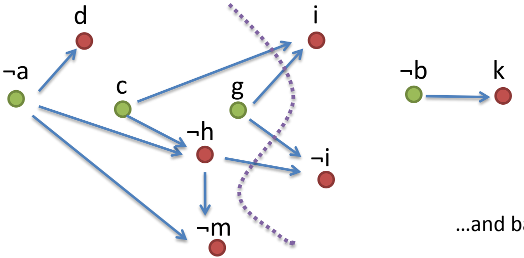


$\neg(c \land g \land \neg h)$

Learning

$$
\begin{aligned}
& (a \lor d) \\
\land\ & (a \lor \neg c \lor \neg h) \\
\land\ & (a \lor h \lor \neg m) \\
\land\ & (b \lor k) \\
\land\ & (\neg g \lor \neg c \lor i) \\
\land\ & (\neg g \lor h \lor \neg i) \\
\land\ & (g \lor h \lor \neg j) \\
\land\ & (g \lor j \lor \neg m)
\end{aligned}
$$





$\neg(c \land g \land \neg h)$

...and backtrack to c, then assert ¬g !

Learning

- Learning has a dramatically positive impact.

- Learning also makes *restarts* possible:
  - Idea: after some number of literal assignments, drop the assignment stack and restart from zero.
  - Goal: avoid locally difficult subtrees.
  - Clauses encode previous knowledge and make new search faster.

Picking Variable Assignments

- Potential strategies:
  - Fixed ordering,
  - Frequency based,
  - “Maximal impact”.

Picking Variable Assignments

- Potential strategies:
  - Fixed ordering,
  - Frequency based,
  - “Maximal impact”.

- Overall favorite are activity-based heuristics:
  - Pick variables that you have seen a lot in conflicts.
  - Decay weights to favor recent conflicts.
  - Cheap to compute/update.

More Engineering…

- SAT dirty little secret: the enormous impact of preprocessing.
  - Problems are generated automatically (“compiled”); many redundancies, symmetry, etc.
  - Preprocessors look for subsumed clauses, equivalent clauses, etc.
  - Typically, run with timeout, then DPLL search.

More Engineering…

- SAT dirty little secret: the enormous impact of preprocessing.
  - Problems are generated automatically (“compiled”); many redundancies, symmetry, etc.
  - Preprocessors look for subsumed clauses, equivalent clauses, etc.
  - Typically, run with timeout, then DPLL search.

- Parallel SAT
  - State-of-the-art is to run instances with different parameters in parallel.

Beyond SAT

- SMT solvers
  - Idea: use a SAT solver for the propositional structure, and theory solvers for conjunction of literals.
- QBF
  - SAT with quantifiers. PSPACE complete.