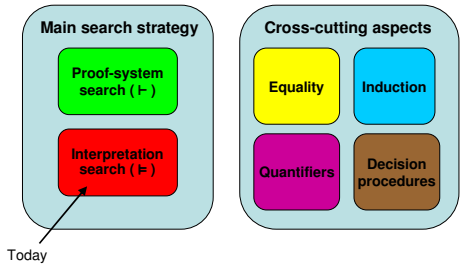
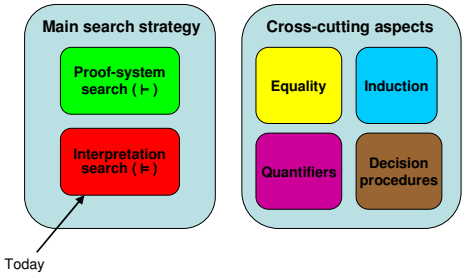
Last time



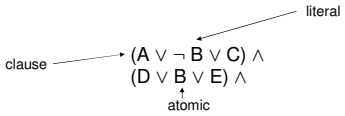
Last time



Search in the semantic domain

Some definitions

- atomic formula: smallest formula possible (no sub-formulas)
- literal: atomic formula or negation of an atomic formula
- clause: disjunction of literals



- CNF: Conjunction of clauses

DPLL backtracking search algorithm

- David-Puttnam-Logemann-Loveland
- Algorithm: given a formula, return SAT or UNSAT
  - SAT: there some truth assignment that makes the formula true
  - UNSAT: formula is false on all truth assignments
- Key idea
  - Pick a literal
  - Assign literal to true, simplify the formula, and recurse
  - Assign literal to false, simplify the formula, and recurse

In more detail

- If formula is false, return UNSAT
- else If formula is true, return SAT
- else:
  - Pick a literal
  - Assign literal to true, simplify the formula, and recurse
    - If recursive call returns SAT, return SAT
  - Assign literal to false, simplify the formula, and recurse
    - If recursive call returns SAT, return SAT
  - If both recursive calls return UNSAT, return UNSAT

Example simplification

(A ∨ ¬ B ∨ C) ∧
(D ∨ B ∨ E) ∧
(¬ A ∨ D ∨ ¬ E)

A to true →

~~(A ∨ ¬ B ∨ C) ∧~~
(D ∨ B ∨ E) ∧
~~(¬ A ∨ D ∨ ¬ E)~~

(A ∨ ¬ B ∨ C) ∧
(D ∨ B ∨ E) ∧
(¬ A ∨ D ∨ ¬ E)

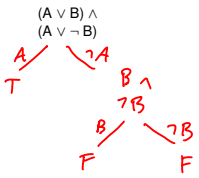
A to false →

~~A~~ ∨ ¬ B ∨ C) ∧
(D ∨ B ∨ E) ∧
~~(¬ A ∨ D ∨ ¬ E)~~

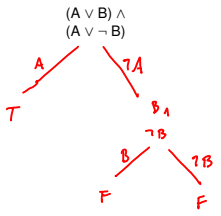
How do formulas become T or F?

- Formula becomes true
  - when conjunction becomes empty
- Formula becomes false
  - when clause becomes empty

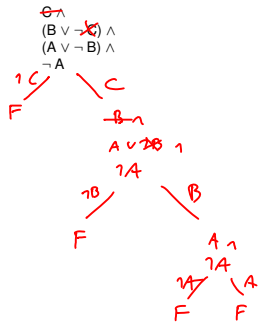
Search tree



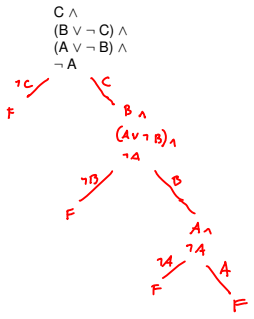
Search tree



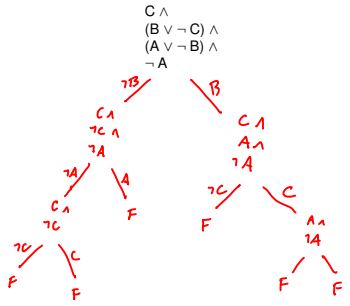
Choice of literal matters



Choice of literal matters



Choice of literal matters



Some heuristics for picking literal

- Pick literals that appear in unit clauses (called unit propagation)

C ∧
(B ∨ ¬ C) ∧
(A ∨ ¬ B) ∧
¬ A

- Pick literals that always appear in the same polarity (A or ¬ A)

(A ∨ B) ∧
(A ∨ ¬ B) ∧
(C ∨ B) ∧
(¬ C ∨ ¬ B)

Why? Because of the following optimization:
- if literal is A, then pick A, don't explore ¬ A branch
- if literal is ¬ A, then pick ¬ A, don't explore A branch

Some heuristics for picking literal

- Pick literals for which the formula can be expressed as (R ∨ A) ∧ (Q ∨ ¬ A) ∧ S
  - Can then merge both subtrees into just one subtree that checks (R ∨ Q) ∧ S
- These are just a few simple heuristics
  - Many other heuristics have been developed
  - Decades of research on this

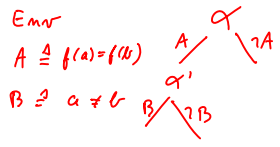
Extending backtracking search

- Let's assume we also have equality with uninterpreted function symbols, for example:

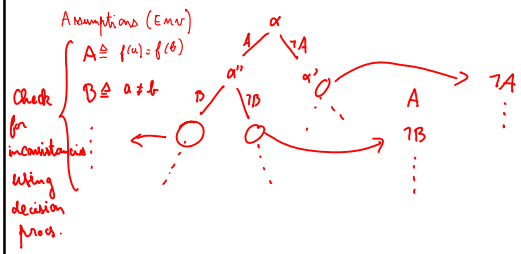
(f(f(a)) = a ∨ ¬ (f(a) = f(b))) ∧
(a = b ∧ f(a) = f(f(b)))

- Some observations
  - We can still simplify a formula based on a literal being T or F
  - But we can only simplify that literal
  - For instance, in the example above, once we've assumed a = b, how do we know that ¬ (f(a) = f(b)) is false?

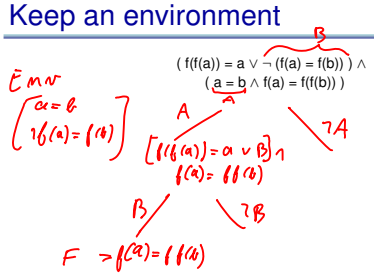
Keep an environment



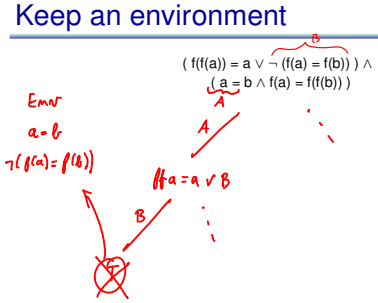
Keep an environment



Keep an environment



Keep an environment



Davis-Putnam paper

- Semi-algorithm for first-order logic
- Refutation based: negation formula, and show that formula is unsatisfiable
- Uses successive SAT instances

Prenex normal form

- Prenex normal form: all quantifiers on the outside
- Some example conversions:
  - $\forall x.\ P(x) \wedge \forall x.\ Q(x)$
  - $\exists x.P(x) \vee \exists x.\ Q(x)$
  - $\forall x.\ P(x) \vee \forall x.\ Q(x)$
- In general can convert any formula into prenex normal form (might possibly strengthen)

Getting rid of existentials

- Replace existential with a function symbol that takes as parameters the enclosing universally quantified variables
- Transform:

  $\forall x_1.\ \exists x_2.\ \forall x_3.\ \exists x_4\ R(x_1, x_2, x_3, x_4)$
- Into

  $\forall x_1.\ \forall x_3.\ R(x_1, f_2(x_1), x_3, f_4(x_1, x_3))$

Herbrand's universe of a formula

- Given a formula F, we call H_F the Herbrand universe of the formula
  - All constants in F belong to H_F (if F does not have constants, then H_F includes a fresh constant a)
  - For any function symbol of arity n occurring in F, and for any $t_1, \ldots, t_n$ belonging to H_F, $f(t_1, \ldots, t_n)$ also belongs to H_F
  - H_F is the minimal set that satisfies these constraints

Quantifier free lines

- Instantiate body of a formula F with elements of H_F
- Suppose $F = \forall x_1, x_2\ R(x_1, f(x_1), x_2)$
- $H_F = \{ a, f(a), f(f(a)), \dots \}$
- Quantifier free lines:
  - R(a, f(a), a)
  - R(a, f(a), f(a))
  - R(f(a), f(f(a)), a)
  - …

- Each line is implied by original formula
- As a result, if the conjunction of some quantifier free lines is inconsistent, so is the original formula

Quantifier free lines

- Each quantifier free line is implied by original formula
- As a result, if the conjunction of some quantifier free lines is inconsistent, so is the original formula
- If the conjunction of the first n quantifier free lines is consistent, for ~~any~~ all n, then the original formula is consistent
  - Follows from the fact that an infinite set of quantifier-free formulas is inconsistent iff some finite subset is inconsistent

Example

- $\forall x. \neg P(x) \vee \exists x. P(x)$

$\exists x. P(x) \wedge \forall x. \neg P(x)$

$\exists x. \forall y. P(x) \wedge \neg P(y)$

$\forall y. P(a) \wedge \neg P(y) \quad H_F = \{a\}$

$P(a) \wedge \neg P(a)$

Example

- $\forall x. \neg P(x) \vee \exists x. P(x)$

Negate: $\exists x. P(x) \wedge \forall x. \neg P(x)$

Rename: $\exists x\ P(x) \wedge \forall y. \neg P(y)$

Prenex: $\exists x \forall y\ .\ P(x) \wedge \neg P(y)$

Herm / Remove $\exists$: $\forall y.\ P(f) \wedge \neg P(y)$

$H = \{f\}$

$P(f) \wedge \neg P(f)$

ATP using Lazy Proof Explication

- $a = b \wedge (\neg (f(a) = f(b)) \vee b = c) \wedge \neg (f(a) = f(c))$

ATP using Lazy Proof Explication

- $a = b \wedge (\neg (f(a) = f(b)) \vee b = c) \wedge \neg (f(a) = f(c))$

x_1 x_2 x_3 x_4

- Assign proxies:
  - $x_1 \wedge (\neg x_2 \vee x_3) \wedge \neg x_4$
- Use SAT solver: if SAT solver says unsatisfiable, then original formula is unsatisfiable

ATP using Lazy Proof Explication

- In this case, say SAT solver comes back with x_1 set to true, and x_2, x_3, and x_4 set to false
- In the propositional world, this is a valid truth assignment
- But when considering the underlying meaning of the proxies, we notice that x_1 being true and x_2 being false is an inconsistency
- If the backtracking search is not aware of this, it will continue considering truth assignments with this same inconsistency (for example $x_1 = x_3 =$ true, $x_2 = x_4 =$ false)

Key idea

- Have decision procedures return an explicating proof as to why the inconsistency occurred.
  - The new formula becomes: F ∧ proof
  - The proof reflects the decision procedure's knowledge back into the propositional world, and can then be used in the prop world to prune the search
- In the example, the proof is:

$$a = b \Rightarrow f(a) = f(b)$$

(x_1 ⇒ x_2)

Example continued

- Formula becomes:

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

- Note that SAT solver cannot find the original satisfying assignment (x_1 set to true, and x_2, x_3, and x_4 set to false)
- Nor can it come back with any assignment that has x_1 set to true and x_2 set to false

Example continued

- So SAT solver comes back with: x_1, x_2, x_3 set to true, and x_4 set to false
- This assignment is also inconsistent when considering the underlying meaning of proxies
- Explicating proof:

$$(a = b \wedge b = c) \Rightarrow f(a) = f(c)$$

(x_1 ∧ x_3 ⇒ x_4)

Example continued

- New formula:

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

- SAT solver returns unsatisfiable, and so we know the original formula is unsatisfiable.

Algorithm in more detail

```
function satisfy(Formula F): Monome {
  while (true)
    "allocate proxy prop vars for atomic formulas in F, and create
    mapping Γ from proxies to atomic formulas"
    TruthAssignment A := SAT-solve(Γ⁻¹(F));
    if (A = null) { // F is unsatisfiable
      return null
    } else
      Monome M := Γ(A);
      Formula E := check(M);
      if (E = null) { // M is satisfiable, then so is F
        return M;
      } else {
        F := F ∧ E;
      }
    }
}
```