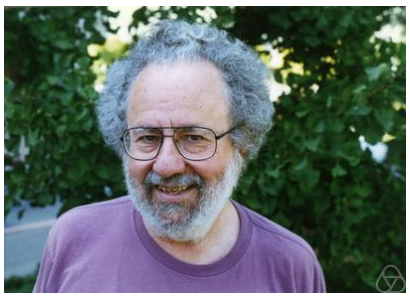
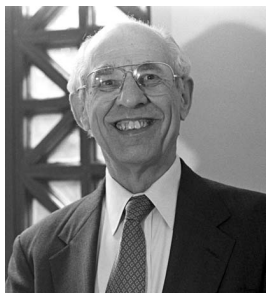
Propositional Logic

DPLL: Davis-Putnam-Logemann-Loveland

Davis–Putnam–Logemann–Loveland

DPLL algorithm:

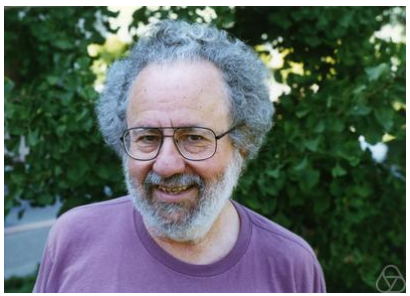
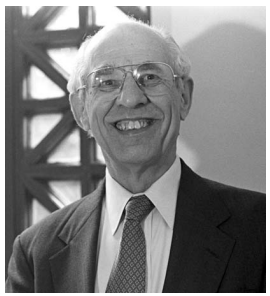
- combines search and deduction to decide satisfiability
- underlies most modern SAT solvers
- is over 50 years old

Davis–Putnam–Logemann–Loveland

DPLL algorithm:

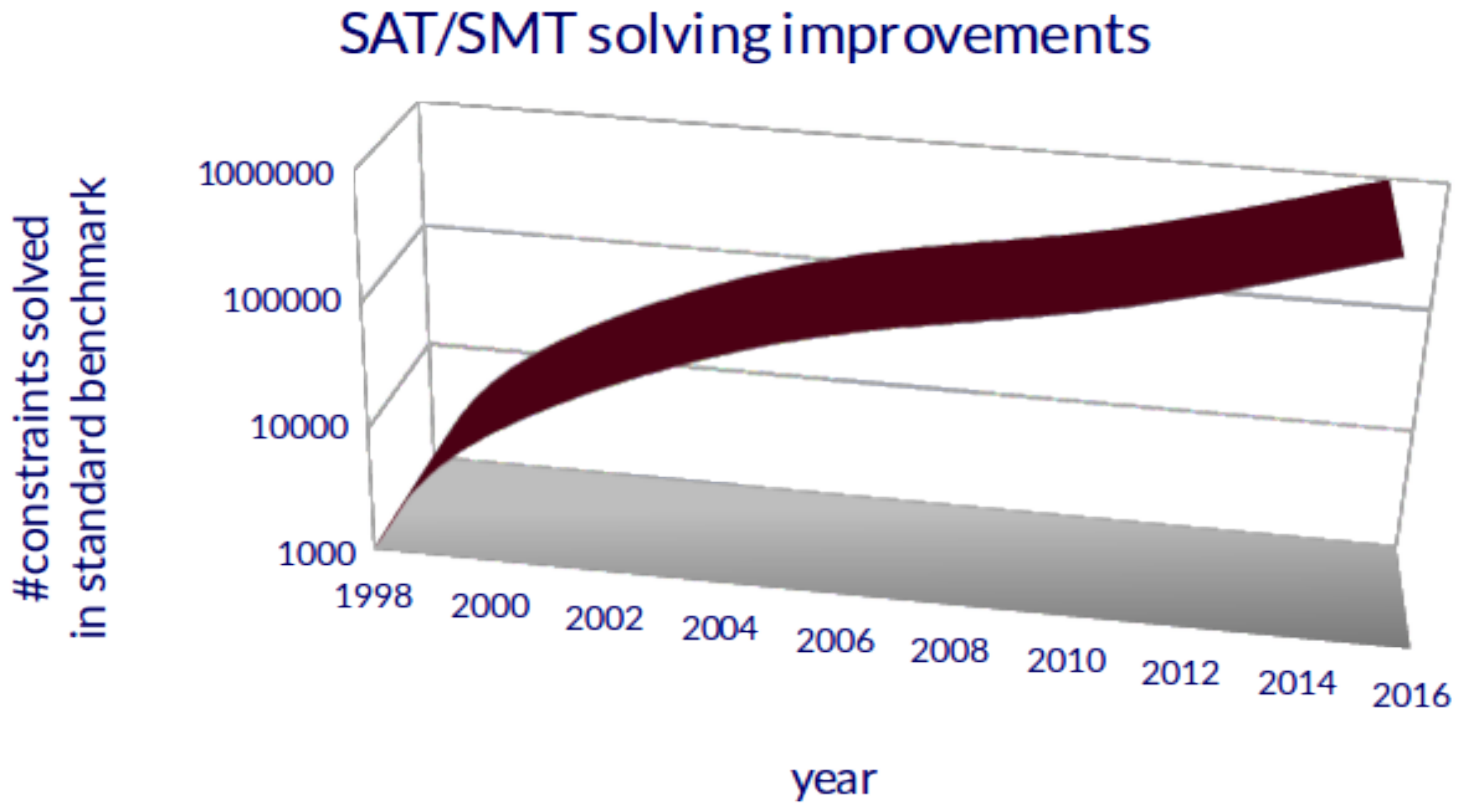
- combines search and deduction to decide satisfiability
- underlies most modern SAT solvers
- is over 50 years old

DPLL-based SAT solvers $\geq$ 1990:

- clause learning
- non-chronological backtracking
- branching heuristics
- lazy evaluation

Performance increase of SAT solvers



Clause representation of CNF formulas

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \ldots \vee L_{1,n_1}) \wedge \ldots \wedge (L_{k,1} \vee \ldots \vee L_{1,n_k})$

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \ldots \vee L_{1,n_1}) \wedge \ldots \wedge (L_{k,1} \vee \ldots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\{\underbrace{\{L_{1,1}, \ldots, L_{1,n_1}\}}_{\textit{clause}}, \ldots, \{L_{k,1}, \ldots, L_{1,n_k}\}\}$$

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \ldots \vee L_{1,n_1}) \wedge \ldots \wedge (L_{k,1} \vee \ldots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\{\underbrace{\{L_{1,1}, \ldots, L_{1,n_1}\}}_{\textit{clause}}, \ldots, \{L_{k,1}, \ldots, L_{1,n_k}\}\}$$

Clause = set of literals (disjunction).

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \ldots \vee L_{1,n_1}) \wedge \ldots \wedge (L_{k,1} \vee \ldots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\{\underbrace{\{L_{1,1}, \ldots, L_{1,n_1}\}}_{\textit{clause}}, \ldots, \{L_{k,1}, \ldots, L_{1,n_k}\}\}$$

Clause = set of literals (disjunction).

Formula in CNF = set of clauses

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \ldots \vee L_{1,n_1}) \wedge \ldots \wedge (L_{k,1} \vee \ldots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\{\underbrace{\{L_{1,1}, \ldots, L_{1,n_1}\}}_{\textit{clause}}, \ldots, \{L_{k,1}, \ldots, L_{1,n_k}\}\}$$

Clause = set of literals (disjunction).

Formula in CNF = set of clauses

Degenerate cases:

The empty clause stands for $\bot$.

Clause representation of CNF formulas

CNF: $(L_{1,1} \vee \ldots \vee L_{1,n_1}) \wedge \ldots \wedge (L_{k,1} \vee \ldots \vee L_{1,n_k})$

Representation as set of sets of literals:

$$\{\underbrace{\{L_{1,1}, \ldots, L_{1,n_1}\}}_{\textit{clause}}, \ldots, \{L_{k,1}, \ldots, L_{1,n_k}\}\}$$

Clause = set of literals (disjunction).

Formula in CNF = set of clauses

Degenerate cases:

The empty clause stands for $\bot$.
The empty set of clauses stands for $\top$.

The joy of sets

We get "for free":

The joy of sets

We get "for free":

- Commutativity:
  $A \vee B \equiv B \vee A$, both represented by $\{A, B\}$

The joy of sets

We get "for free":

- Commutativity:
  $A \vee B \equiv B \vee A$, both represented by $\{A, B\}$
- Associativity:
  $(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$

The joy of sets

We get "for free":

- Commutativity:
  $A \vee B \equiv B \vee A$, both represented by $\{A, B\}$
- Associativity:
  $(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$
- Idempotence:
  $(A \vee A) \equiv A$, both represented by $\{A\}$

The joy of sets

We get "for free":

- Commutativity:
  $A \vee B \equiv B \vee A$, both represented by $\{A, B\}$
- Associativity:
  $(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$
- Idempotence:
  $(A \vee A) \equiv A$, both represented by $\{A\}$

Sets are a convenient representation of conjunctions and disjunctions with built in associativity, commutativity and idempotence

CNF-SAT: Input: Set of clauses F

The joy of sets

We get "for free":

- Commutativity:
  $A \vee B \equiv B \vee A$, both represented by $\{A, B\}$
- Associativity:
  $(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$
- Idempotence:
  $(A \vee A) \equiv A$, both represented by $\{A\}$

Sets are a convenient representation of conjunctions and disjunctions with built in associativity, commutativity and idempotence

CNF-SAT: Input: Set of clauses F
Question: Is F unsatisfiable?

The joy of sets

We get "for free":

- Commutativity:
  $A \vee B \equiv B \vee A$, both represented by $\{A, B\}$
- Associativity:
  $(A \vee B) \vee C \equiv A \vee (B \vee C)$, both represented by $\{A, B, C\}$
- Idempotence:
  $(A \vee A) \equiv A$, both represented by $\{A\}$

Sets are a convenient representation of conjunctions and disjunctions with built in associativity, commutativity and idempotence

CNF-SAT: Input: Set of clauses F
Question: Is F unsatisfiable?

DPLL — The simplest algorithm for CNF-SAT

Simplest algorithm: Construct the truth table.
Best-case runtime is $\Theta(m \cdot 2^n)$ for a formula of length m over n variables.

DPLL — A first improvement: Partial Evaluation

Improvement: partial evaluation using Boole-Shannon expansion

Lemma (Boole-Shannon Expansion)

For every formula F and atom A:

$$F \equiv (A \wedge F[\top/A]) \vee (\neg A \wedge F[\bot/A]).$$

Proof By structural induction on F (exercise).

Corollary

F is satisfiable iff $F[\bot/A]$ or $F[\top/A]$ are satisfiable.

DPLL — First step: partial evaluation

$F[\bot/A]$ and $F[\top/A]$ easy to compute in clause normal form:

$F[\top/A] \equiv$ take F, remove all clauses with A, remove all $\neg A$.
$F[\bot/A] \equiv$ take F, remove all clauses with $\neg A$, remove all A.

DPLL — First step: partial evaluation

$F[\bot/A]$ and $F[\top/A]$ easy to compute in clause normal form:

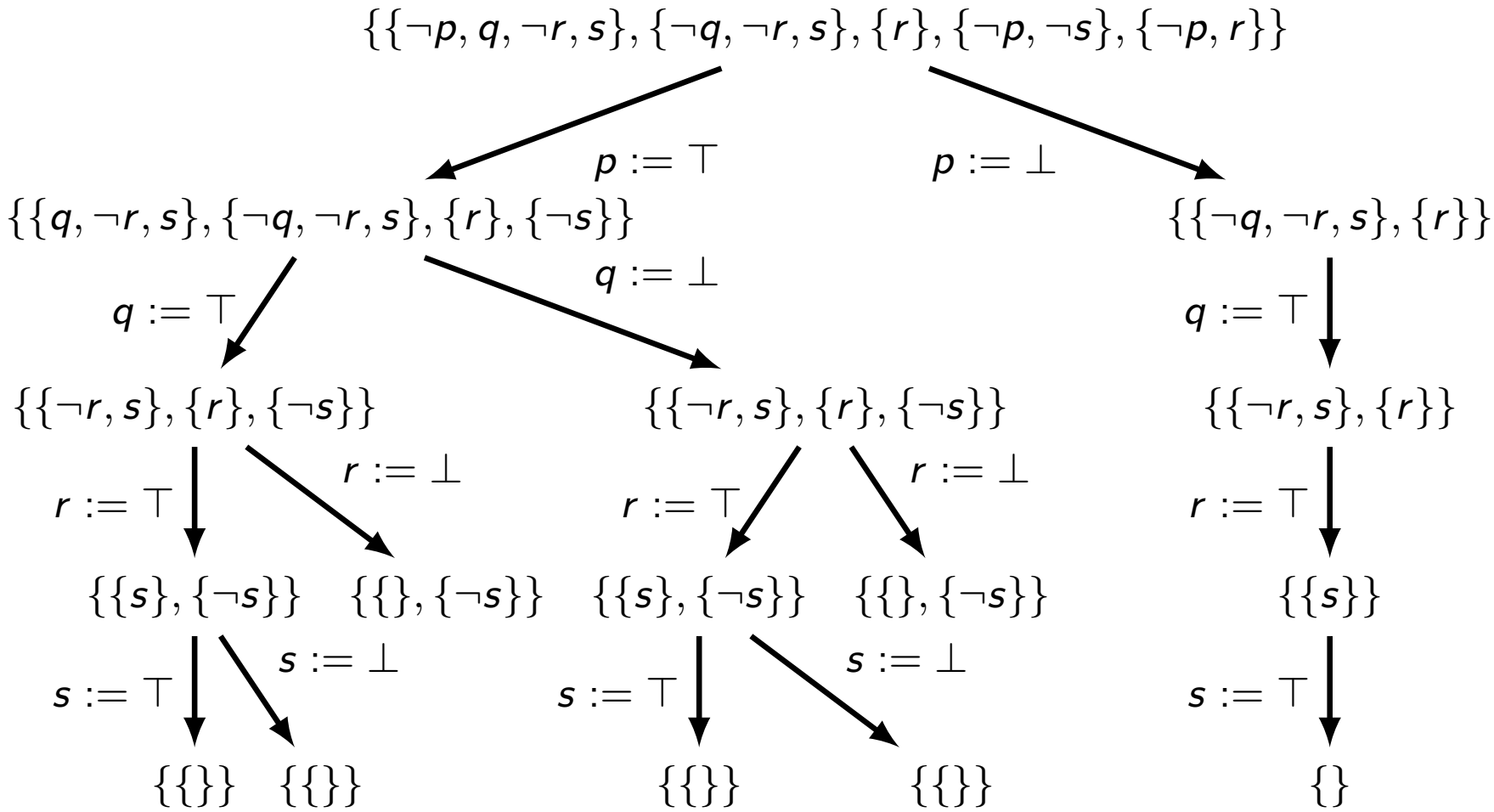
$F[\top/A] \equiv$ take F, remove all clauses with A, remove all $\neg A$.
$F[\bot/A] \equiv$ take F, remove all clauses with $\neg A$, remove all A.

Partial evaluation algorithm:

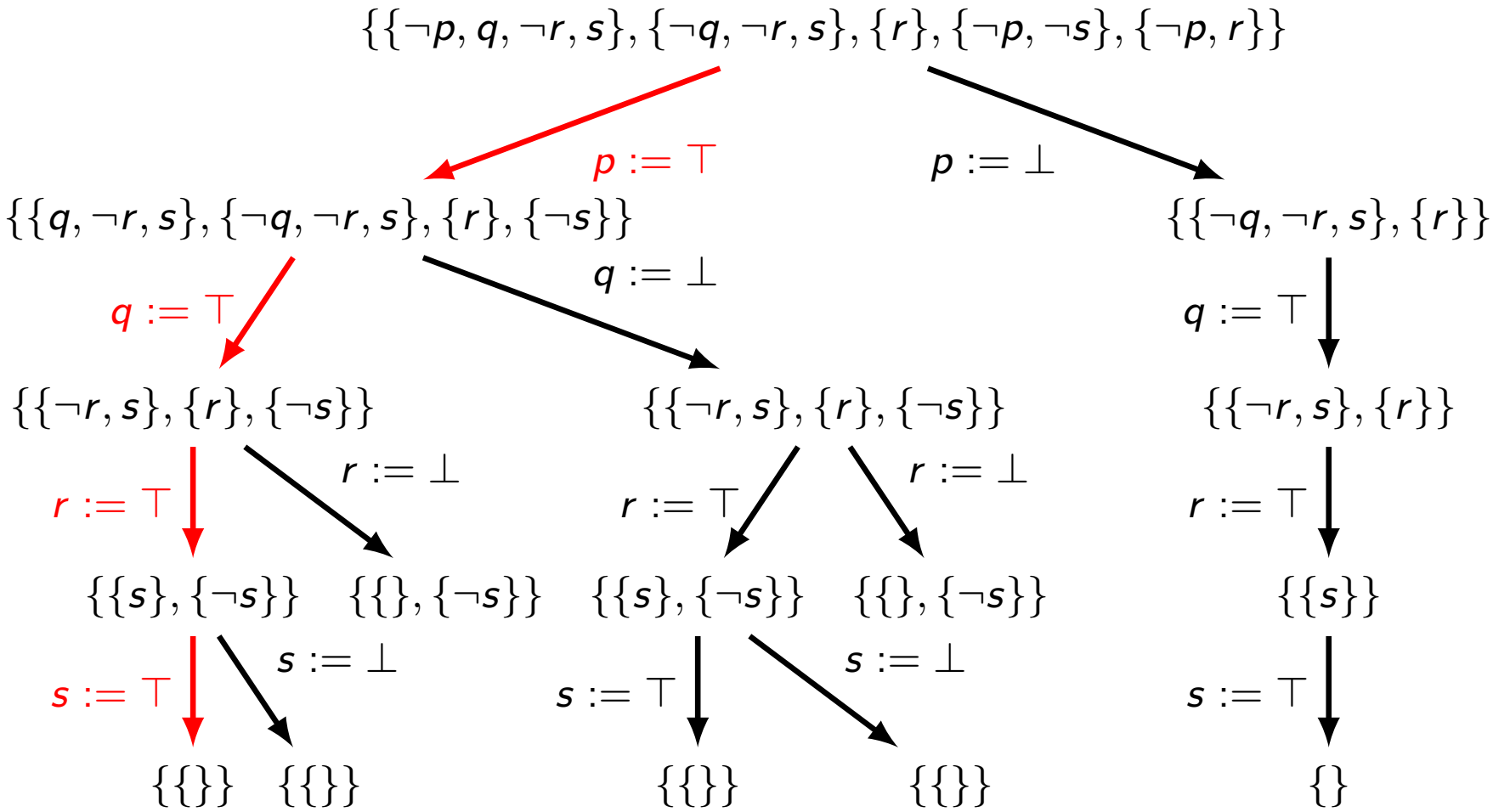
Given formula F, total order on the variables $\prec$:
If $\{\} \in F$ return unsatisfiable.
If $F = \emptyset$ return satisfiable.
Otherwise:

- Fix the first variable A in F according to $\prec$.
- Recursively check if $F[\bot/A]$ is satisfiable;
- if yes, return satisfiable.
- Recursively check if $F[\top/A]$ is satisfiable;
- if yes, return satisfiable, otherwise unsatisfiable.

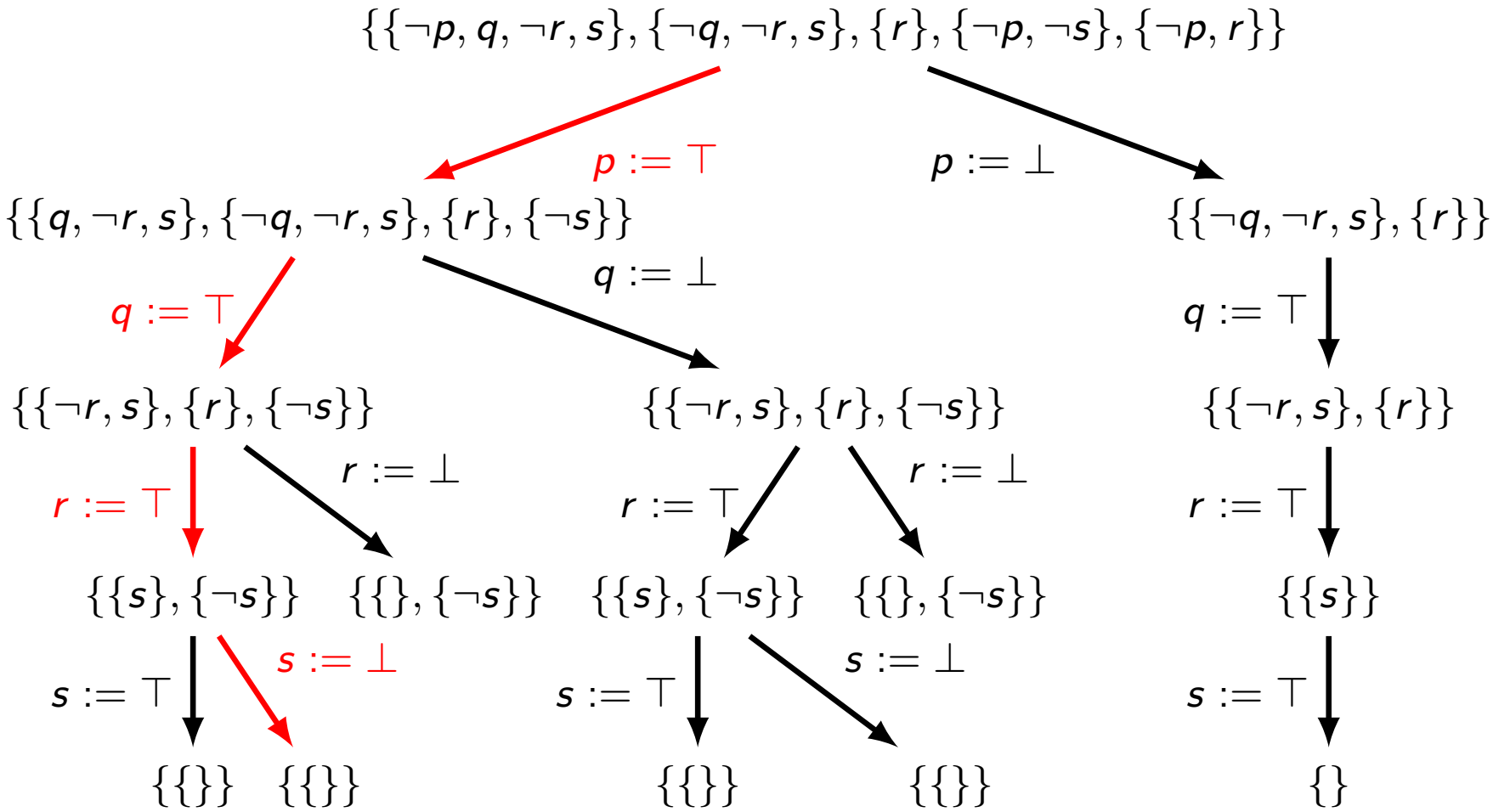
DPLL: Davis-Putnam-Logemann-Loveland



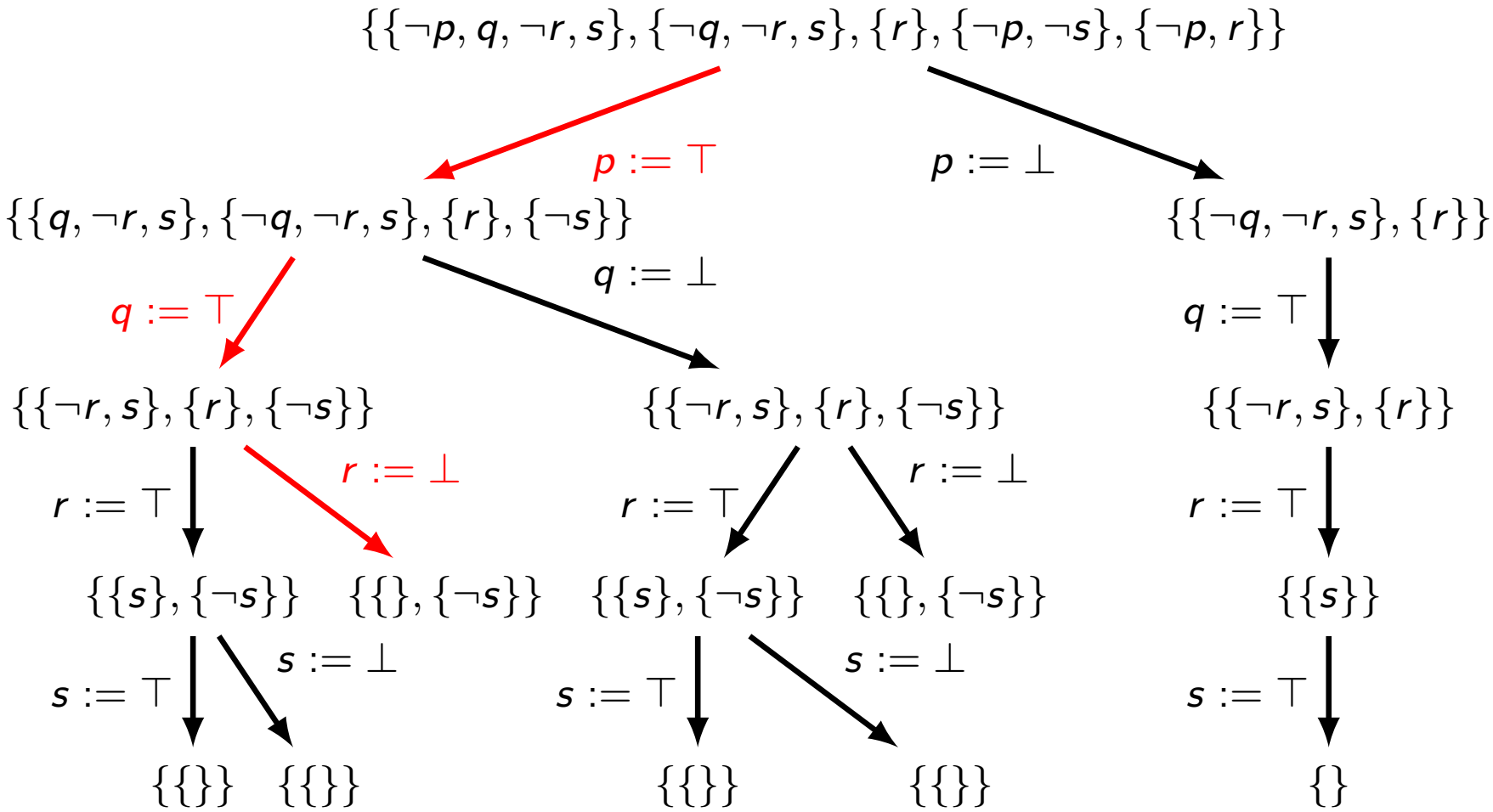
DPLL: Davis-Putnam-Logemann-Loveland



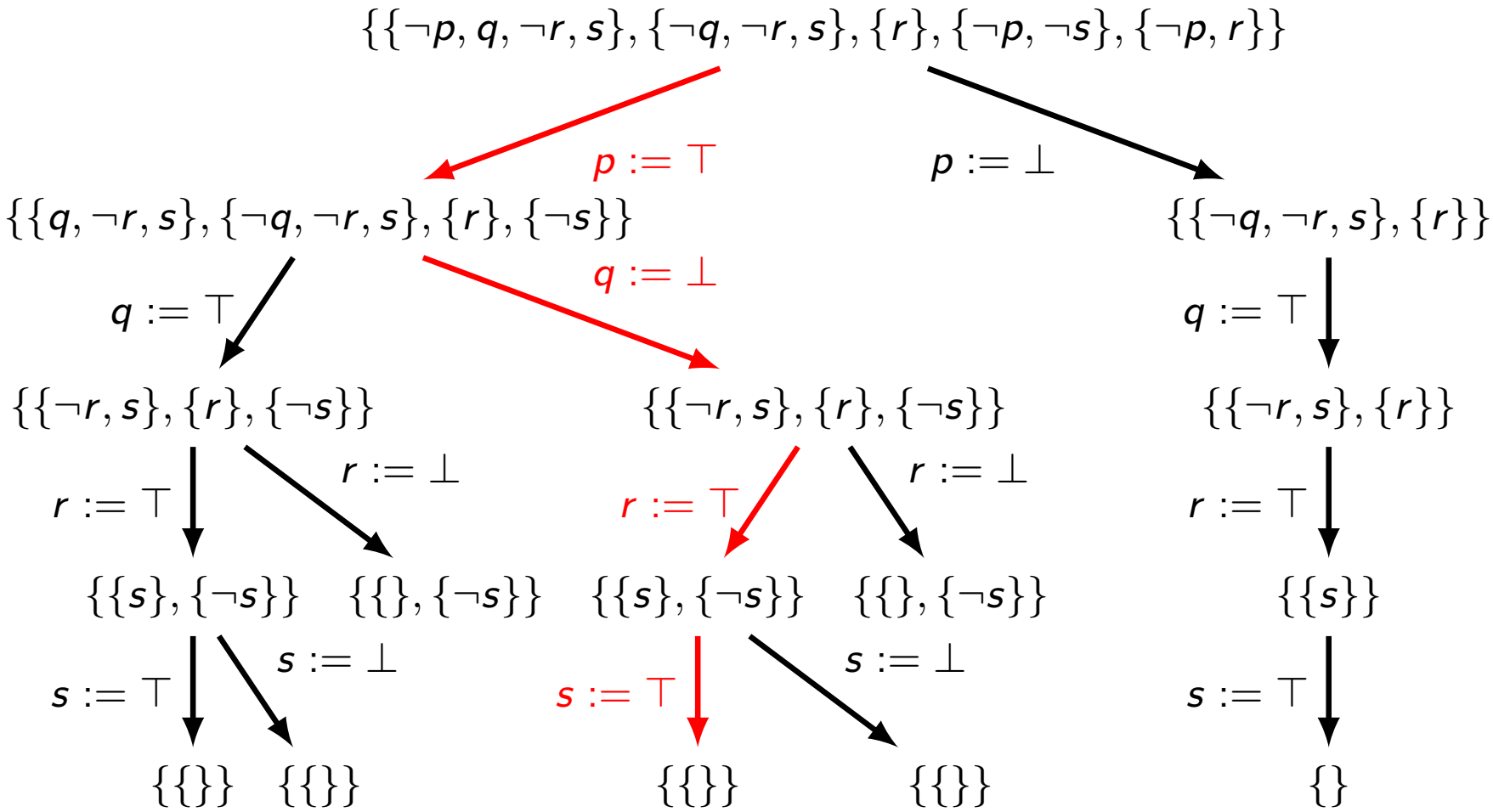
DPLL: Davis-Putnam-Logemann-Loveland



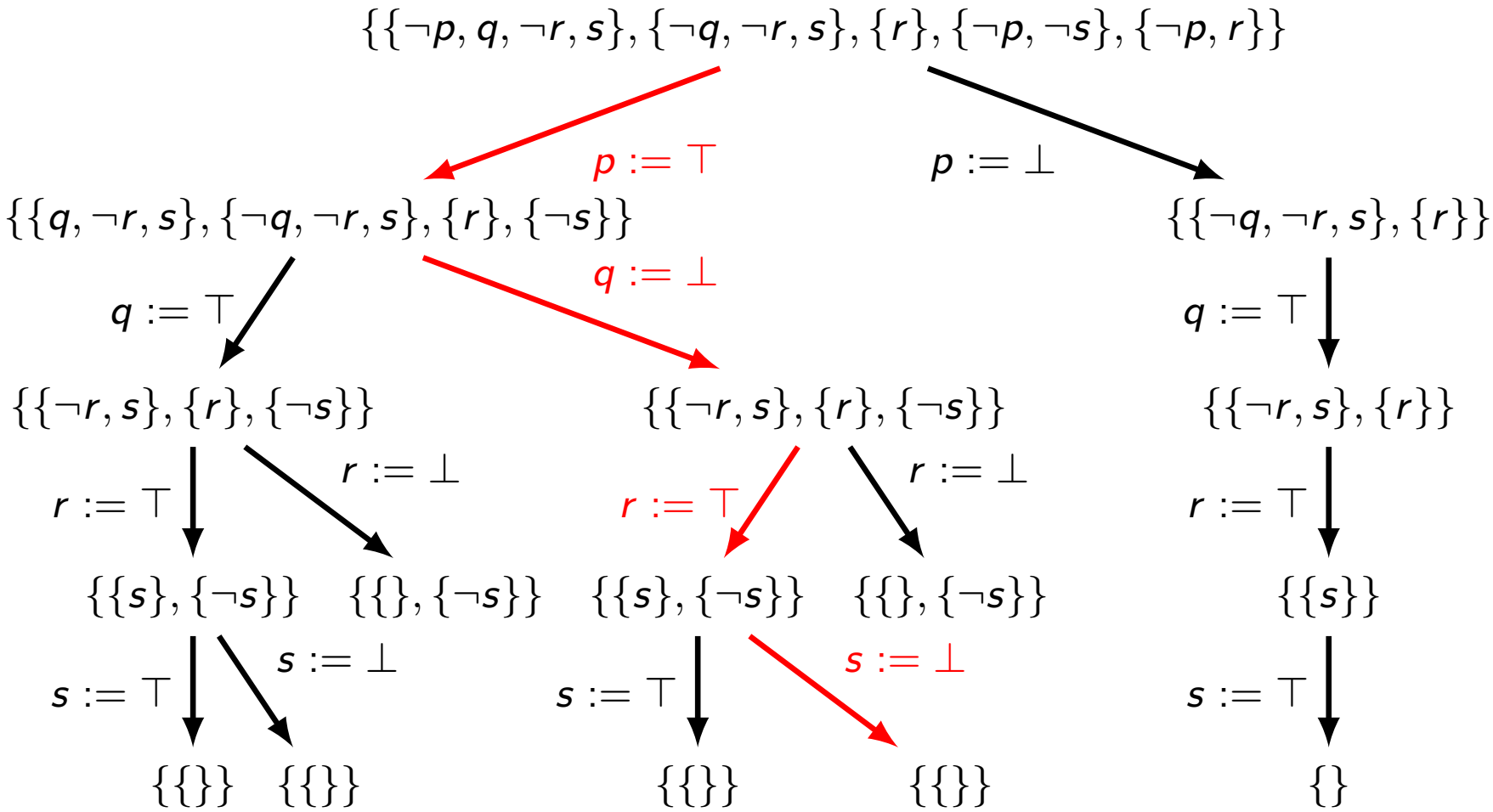
DPLL: Davis-Putnam-Logemann-Loveland



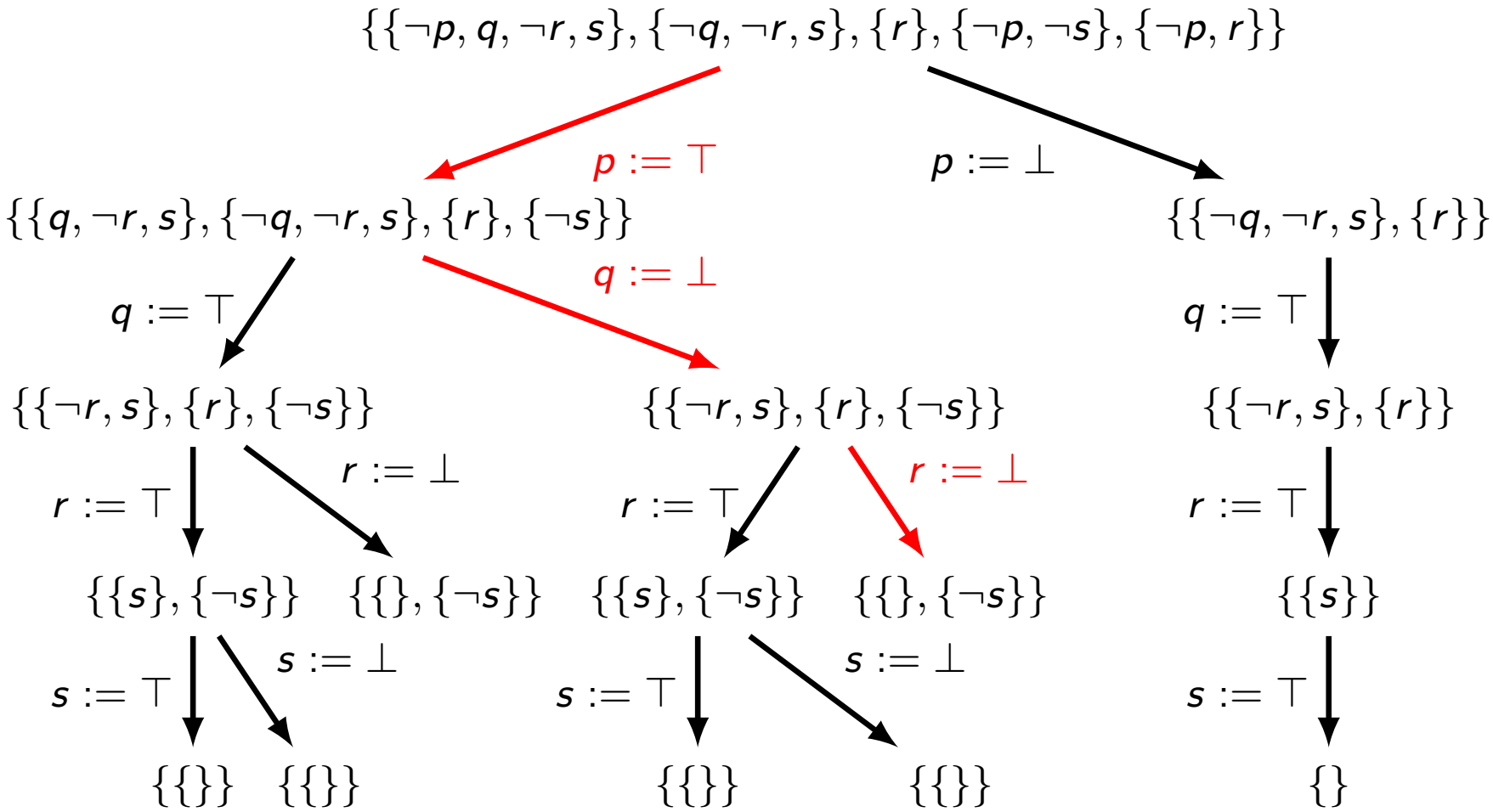
DPLL: Davis-Putnam-Logemann-Loveland



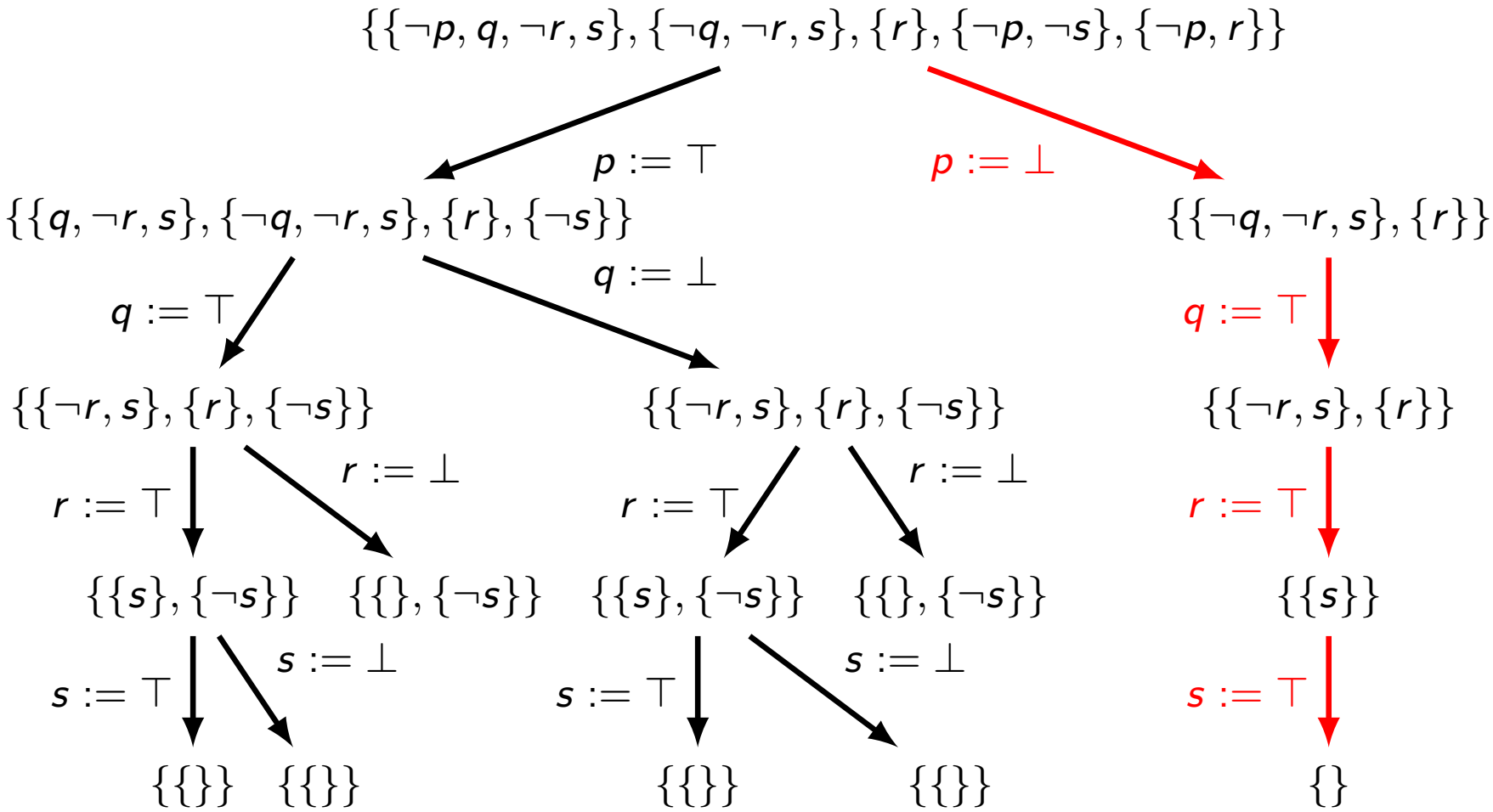
DPLL: Davis-Putnam-Logemann-Loveland



DPLL: Davis-Putnam-Logemann-Loveland



DPLL: Davis-Putnam-Logemann-Loveland



DPLL: Davis-Putnam-Logemann-Loveland

Instead of fixing an order on variables, choose the next variable dynamically.

- OLR: one-literal rule If $\{L\} \in F$ ($\{L\}$ is called unit clause), then every satisfying assignment sets L to true. So it suffices to check satisfiability of $F[\top/L]$.
- PLR: pure-literal rule
  If L appears in F and $\overline{L}$ does not, then it also suffices to check satisfiability of $F[\top/L]$ (Why?).

DPLL algorithm: Partial evaluation that gives priority to a variable satisfying OLR, then to a variable satisfying PLR, and otherwise picks the first unpicked variable of $\prec$.

Applying OLR can generate further unit clauses (unit propagation). Same for PLR, but DPLL often implemented with only OLR for efficiency.

DPLL: Davis-Putnam-Logemann-Loveland

$$\{\{\neg p, q, \neg r, s\}, \{\neg q, \neg r, s\}, \{r\}, \{\neg p, \neg s\}, \{\neg p, r\}\}$$

$$\downarrow \text{OLR } r := \top$$

$$\{\{\neg p, q, s\}, \{\neg q, s\}, \{\neg p, \neg s\}\}$$

$$\downarrow \text{PLR } p := \bot$$

$$\{\{\neg q, s\}\}$$

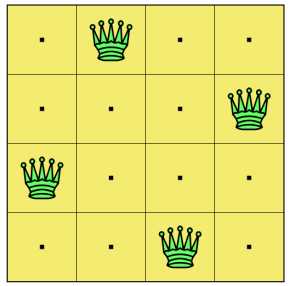
$$\downarrow \text{PLR } q := \bot$$

$$\{\}$$

In this example PLR and OLR allow us to avoid all case splits.

Example: 4 queens

Problem: place 4 non-attacking queens on a 4x4 chess board



Variable p_{ij} models: there is a queen in square (i, j)

- ≥ 1 in each row: $\bigwedge_{i=1}^{4} \bigvee_{j=1}^{4} p_{ij}$
- ≤ 1 in each row: $\bigwedge_{i=1}^{4} \bigwedge_{j \neq j'=1}^{4} \neg p_{ij} \vee \neg p_{ij'}$
- ≤ 1 in each column: $\bigwedge_{j=1}^{4} \bigwedge_{i \neq i'=1}^{4} \neg p_{ij} \vee \neg p_{i'j}$
- ≤ 1 on each diagonal: $\bigwedge_{i,j=1}^{4} \bigvee_{k} \neg p_{i-k,i+k} \vee \neg p_{i+k,j+k}$

Total number of clauses: $4 + 24 + 24 + 28 = 80$

DPLL: 4 queens

Running the DPLL algorithm:

- Start with $p_{11} \mapsto 1$
  delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, delete $\neg p_{11}$: 9 new unit clauses
  unit propagation: deletes 65 clauses!

DPLL: 4 queens

Running the DPLL algorithm:

- Start with $p_{11} \mapsto 1$
  delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, delete $\neg p_{11}$: 9 new unit clauses
  unit propagation: deletes 65 clauses!

- Set $p_{23} \mapsto 1$
  4 new unit clauses: $\{\neg p_{24}\}, \{\neg p_{43}\}, \{\neg p_{32}\}, \{\neg p_{34}\}$
  unit propagation of $\{\neg p_{34}\}$: UNSAT

DPLL: 4 queens

Running the DPLL algorithm:

- Start with $p_{11} \mapsto 1$
  delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, delete $\neg p_{11}$: 9 new unit clauses
  unit propagation: deletes 65 clauses!

- Set $p_{23} \mapsto 1$
  4 new unit clauses: $\{\neg p_{24}\}, \{\neg p_{43}\}, \{\neg p_{32}\}, \{\neg p_{34}\}$
  unit propagation of $\{\neg p_{34}\}$: UNSAT

  fixing only two literals collapsed from 80 clauses to 1
  ruled out 2^{14} of 2^{16} possible assignments!

- Backtrack: $p_{11} \mapsto 0$, $p_{12} \mapsto 1$
  delete $\{\neg p_{12}\}$: 9 new unit clauses
  unit propagation: leaves only 1 clause $\{p_{43}\}$!

DPLL: 4 queens

Running the DPLL algorithm:

- Start with $p_{11} \mapsto 1$
  delete $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, delete $\neg p_{11}$: 9 new unit clauses
  unit propagation: deletes 65 clauses!

- Set $p_{23} \mapsto 1$
  4 new unit clauses: $\{\neg p_{24}\}, \{\neg p_{43}\}, \{\neg p_{32}\}, \{\neg p_{34}\}$
  unit propagation of $\{\neg p_{34}\}$: UNSAT

  fixing only two literals collapsed from 80 clauses to 1
  ruled out 2^{14} of 2^{16} possible assignments!

- Backtrack: $p_{11} \mapsto 0$, $p_{12} \mapsto 1$
  delete $\{\neg p_{12}\}$: 9 new unit clauses
  unit propagation: leaves only 1 clause $\{p_{43}\}$!

- Answer: $p_{12}, p_{24}, p_{31}, p_{43} \mapsto 1$

DPLL: Evaluation

Oriented towards satisfiability:

- $2^{O(n)}$ time for satisfiable formulas, but $2^{\Theta(n)}$ for unsatisfiable ones.
- DPLL computes a satisfying assignment, if there is one.
- The satisfying assignment is a certificate of satisfiability.
- Satisfiable formulas have short certificates: satisfying assignment never larger than the formula.

Coming next: resolution, a procedure oriented towards unsatisfiability.

- $2^{O(n)}$ time for unsatisfiable formulas, but $2^{\Theta(n)}$ for satisfiable ones.
- Resolution computes a certificate of unsatisfiabiity.
- However, the certificate is exponentially longer than the formula in the worst case.
- Polynomial certificates for satisfiability implies NP$=$ coNP.