FDPLL – A First-Order Davis-Putnam-Logemann-Loveland Procedure

Peter Baumgartner
Institut für Informatik
Universität Koblenz-Landau
Germany

http://www.uni-koblenz.de/~peter/

Theorem Proving

Theorem proving is about . . .

- Logics (Propositional, First-Order, Higher-Order, Modal, Description, . . .)
- Calculi and proof procedures (Resolution,. . .)
- Systems (Interactive, Automated)
- Applications (Knowledge Representation, Verification, . . .)

Theorem Proving

Theorem proving is about ...

Logics (Propositional, First-Order, Higher-Order, Modal, Description, ...)

Calculi and proof procedures (Resolution,...)

Systems (Interactive, Automated)

Applications (Knowledge Representation, Verification, ...)

Milestones

60s: Calculi: DPLL, Resolution, Model Elimination

70s: Logic Programming

80s: Knowledge Representation

90s: "A Basis for Applications"

2000s: Semantic Web, Ontologies, SW-Engineering

Theorem Proving

Theorem proving is about . . .

Logics (Propositional, First-Order, Higher-Order, Modal, Description, . . .)

Calculi and proof procedures (Resolution, . . .)

Systems (Interactive, Automated)

Applications (Knowledge Representation, Verification, . . .)

Milestones

60s: Calculi: DPLL, Resolution, Model Elimination

70s: Logic Programming

80s: Knowledge Representation

90s: "A Basis for Applications"

2000s: Semantic Web, Ontologies, SW-Engineering

Two Separated Worlds

	First-Order Reasoning	**Propositional Reasoning**
Techniques	Resolution	DPLL
	Model Elimination	OBDD
	Hyper Linking	Stalmarck's Method
		Tableaux
		Stochastic (GSAT)
Systems	E, Otter, Setheo, SNARK, Spass, Vampire	Chaff, SMV, Heerhugo, FACT, WalkSat
Applications	SW-Verification (Limited)	Symbolic Model Checking
	Mathematics	Mathematics
	Discourse Representation	Planning, Description Logics
	TPTP	Nonmonotonic Reasoning

Two Separated Worlds

	First-Order Reasoning	Propositional Reasoning
Techniques	Resolution	DPLL
	Model Elimination	OBDD
	Hyper Linking	Stalmarck's Method
		Tableaux
		Stochastic (GSAT)
Systems	E, Otter, Setheo, SNARK, Spass, Vampire	Chaff, SMV, Heerhugo, FACT, WalkSat
Applications	SW-Verification (Limited)	Symbolic Model Checking
	Mathematics	Mathematics
	Discourse Representation	Planning, Description Logics
	TPTP	Nonmonotonic Reasoning

Can couple these worlds more closely?

Motivation

DPLL: Successfully used for propositional logic

FDPLL: New lifting of DPLL to **f**irst-order logic

Why?

Motivation

DPLL: Successfully used for propositional logic

FDPLL: New lifting of DPLL to **f**irst-order logic

Why?

- Use successful first-order techniques (unification, redundancy tests)

Unification:

$$\mathrm{unify}\{P(a,y),\ P(x,f(x))\}$$
$$= \{P(a,f(a))\}$$

Motivation

DPLL: Successfully used for propositional logic

FDPLL: New lifting of DPLL to **f**irst-order logic

Why?

- Use successful first-order techniques (unification, redundancy tests)
- Close a gap in the calculus landscape

Unification:

$$\text{unify}\{\mathrm{P}(\mathrm{a},\mathrm{y}),\ \mathrm{P}(\mathrm{x},\mathrm{f}(\mathrm{x}))\}$$
$$= \{\mathrm{P}(\mathrm{a},\mathrm{f}(\mathrm{a}))\}$$

Motivation

DPLL: Successfully used for propositional logic

FDPLL: New lifting of DPLL to **f**irst-order logic

Why?

- Use successful first-order techniques (unification, redundancy tests)
- Close a gap in the calculus landscape
- Theorem Proving: Alternative to Resolution, Model Elimination

Unification:

$$\mathrm{unify}\{\mathrm{P}(\mathrm{a},\mathrm{y}),\ \mathrm{P}(\mathrm{x},\mathrm{f}(\mathrm{x}))\}$$
$$= \{\mathrm{P}(\mathrm{a},\mathrm{f}(\mathrm{a}))\}$$

Theorem Proving:

$$\text{Axioms} \overset{?}{\models} \text{Conjecture}$$

Motivation

DPLL: Successfully used for propositional logic

FDPLL: New lifting of DPLL to **f**irst-order logic

Why?

- Use successful first-order techniques (unification, redundancy tests)
- Close a gap in the calculus landscape
- Theorem Proving: Alternative to Resolution, Model Elimination
- Model computation (Counterexamples, diagnosis, abduction, planning, nonmonotonic reasoning,...– largely unexplored)

Unification:

$$\text{unify}\{P(a,y),\ P(x,f(x))\}$$
$$= \{P(a,f(a))\}$$

Theorem Proving:

$$\text{Axioms} \overset{?}{\models} \text{Conjecture}$$

Model Computation: Is

$$\text{Axioms} \land \neg\text{Conjecture}$$

satisfiable?

$$\text{Axioms} \overset{?}{\not\models} \text{Conjecture}$$

Motivation

DPLL: Successfully used for propositional logic
FDPLL: New lifting of DPLL to **f**irst-order logic

Why?

- Use successful first-order techniques (unification, redundancy tests)
- Close a gap in the calculus landscape
- Theorem Proving: Alternative to Resolution, Model Elimination
- Model computation
  (Counterexamples, diagnosis, abduction, planning, nonmonotonic reasoning,... – largely unexplored)
- (Dream) Bring first-order reasoning to domains that are successfully tackled with propositional DPLL

Unification:

$$\text{unify}\{P(a,y),\ P(x,f(x))\}$$
$$= \{P(a,f(a))\}$$

Theorem Proving:

$$\text{Axioms} \stackrel{?}{\models} \text{Conjecture}$$

Model Computation: Is

$$\text{Axioms} \land \neg\text{Conjecture}$$

satisfiable?

$$\text{Axioms} \stackrel{?}{\not\models} \text{Conjecture}$$

Overview

Propositional DPLL as a semantic tree method

First-Order DPLL so far

FDPLL

Relation to other calculi

Notation

Propositional clause: a disjunction of literals, e.g.

$$A \lor B \lor \neg C \lor \neg D$$

Propositional clause set: a finite set of propositional clauses.

Notation

Propositional clause: a disjunction of literals, e.g.

$$A \vee B \vee \neg C \vee \neg D$$

Propositional clause set: a finite set of propositional clauses.

Interpretation: maps atoms to $\{\text{true}, \text{false}\}$, e.g.

A	B	C	D
true	false	true	false

Representation by consistent sets of literals, e.g. (all the same)

$$\{A, C\} \qquad \{A, \neg B, C\} \qquad \{A, \neg B, C, \neg D\}$$

Notation

Propositional clause: a disjunction of literals, e.g.

$$A \lor B \lor \neg C \lor \neg D$$

Propositional clause set: a finite set of propositional clauses.

Interpretation: maps atoms to $\{\mathrm{true}, \mathrm{false}\}$, e.g.

A	B	C	D
true	false	true	false

Representation by consistent sets of literals, e.g. (all the same)

$$\{A, C\} \qquad \{A, \neg B, C\} \qquad \{A, \neg B, C, \neg D\}$$

Model: an interpretation such that every clause is satisfied, e.g.

$$\{A, C\} \models \{A \lor B \lor \neg C \lor \neg D\}$$

$$\{A, C\} \not\models \{A \lor B \lor \neg C \lor \neg D,\ \neg A \lor B\}$$

A clause set is satisfiable iff a model for it exists, otherwise unsatisfiable.

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$

$\langle$empty tree$\rangle$

$\{\} \not\models A \vee B$

$\{\} \models C \vee \neg A$

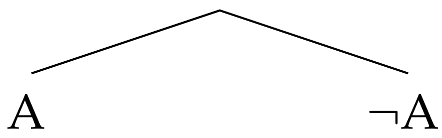
$\{\} \models D \vee \neg C \vee \neg A$

$\{\} \models \neg D \vee \neg B$

- A Branch stands for an interpretation
- Purpose of splitting: Satisfy a clause that is currently "false"
- Close branch if some clause plainly contradicts it ($\star$)

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$



$\{A\} \models A \vee B$

$\{A\} \not\models C \vee \neg A$

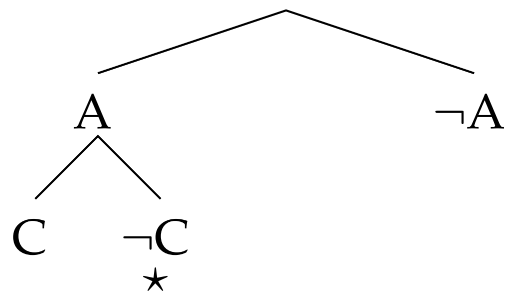
$\{A\} \models D \vee \neg C \vee \neg A$

$\{A\} \models \neg D \vee \neg B$

- A Branch stands for an interpretation
- Purpose of splitting: Satisfy a clause that is currently "false"
- Close branch if some clause plainly contradicts it ($\star$)

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$



$\{A, C\} \models A \vee B$

$\{A, C\} \models C \vee \neg A$

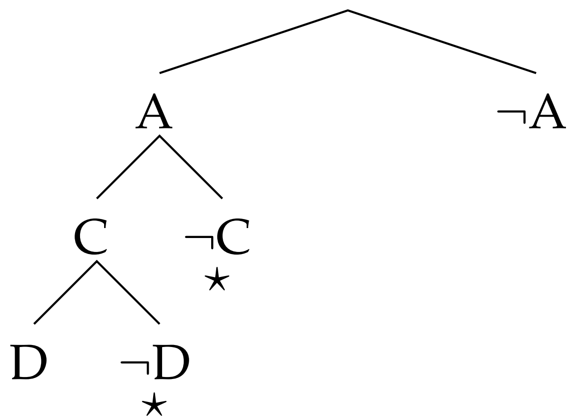
$\{A, C\} \not\models D \vee \neg C \vee \neg A$

$\{A, C\} \models \neg D \vee \neg B$

- A Branch stands for an interpretation
- Purpose of splitting: Satisfy a clause that is currently "false"
- Close branch if some clause plainly contradicts it ($\star$)

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$



$\{A, C, D\} \models A \vee B$

$\{A, C, D\} \models C \vee \neg A$

$\{A, C, D\} \models D \vee \neg C \vee \neg A$

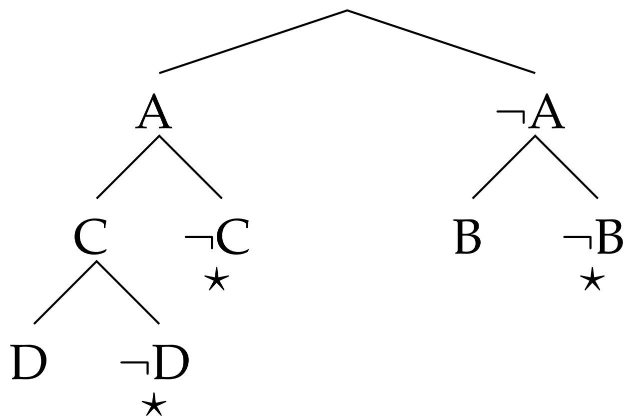
$\{A, C, D\} \models \neg D \vee \neg B$

Model $\{A, C, D\}$ found.

- A Branch stands for an interpretation
- Purpose of splitting: Satisfy a clause that is currently "false"
- Close branch if some clause plainly contradicts it ($\star$)

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$



$\{B\} \models A \vee B$

$\{B\} \models C \vee \neg A$

$\{B\} \models D \vee \neg C \vee \neg A$

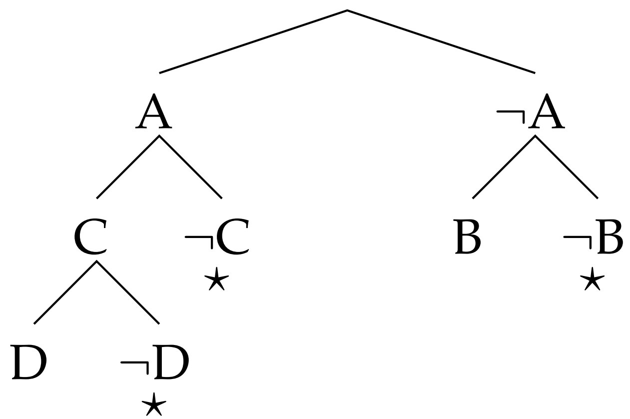
$\{B\} \models \neg D \vee \neg B$

Model $\{B\}$ found.

- A Branch stands for an interpretation
- Purpose of splitting: Satisfy a clause that is currently "false"
- Close branch if some clause plainly contradicts it ($\star$)

Propositional DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$



$\{B\} \models A \vee B$

$\{B\} \models C \vee \neg A$

$\{B\} \models D \vee \neg C \vee \neg A$

$\{B\} \models \neg D \vee \neg B$

Model $\{B\}$ found.

- A Branch stands for an interpretation
- Purpose of splitting: Satisfy a clause that is currently "false"
- Close branch if some clause plainly contradicts it ($\star$)
- **Sound and complete, also for (minimal) model reasoning**

DP vs. DPLL

Two versions of the main inference rule:

Davis, Putnam 1960: "Rule for eliminating atomic formulas":

1. Select an atom A
2. Resolve (!) on all clauses $\mathrm{A} \vee \ldots$ and $\neg\mathrm{A} \vee \ldots$
3. Delete all clauses $\mathrm{A} \vee \ldots$ and $\neg\mathrm{A} \vee \ldots$

DP vs. DPLL

Two versions of the main inference rule:

Davis, Putnam 1960: "Rule for eliminating atomic formulas":

1. Select an atom A
2. Resolve (!) on all clauses $A \vee \ldots$ and $\neg A \vee \ldots$
3. Delete all clauses $A \vee \ldots$ and $\neg A \vee \ldots$

Problem: Step 2 involves multiplying out $\bigvee \bigwedge$-formula to $\bigwedge \bigvee$-formula

DP vs. DPLL

Two versions of the main inference rule:

Davis, Putnam 1960: "Rule for eliminating atomic formulas":

1. Select an atom A
2. Resolve (!) on all clauses $\mathrm{A} \vee \ldots$ and $\neg\mathrm{A} \vee \ldots$
3. Delete all clauses $\mathrm{A} \vee \ldots$ and $\neg\mathrm{A} \vee \ldots$

Problem: Step 2 involves multiplying out $\bigvee\bigwedge$-formula to $\bigwedge\bigvee$-formula
Solution:

Davis, Logemann, Loveland 1962: "Splitting Rule":

1. Select an atom A
2. Split into cases A and $\neg\mathrm{A}$.
3. In each case, simplify according to new information.

DP vs. DPLL

Two versions of the main inference rule:

Davis, Putnam 1960: "Rule for eliminating atomic formulas":

1. Select an atom A
2. Resolve (!) on all clauses $\mathrm{A} \vee \ldots$ and $\neg\mathrm{A} \vee \ldots$
3. Delete all clauses $\mathrm{A} \vee \ldots$ and $\neg\mathrm{A} \vee \ldots$

Problem: Step 2 involves multiplying out $\bigvee\bigwedge$-formula to $\bigwedge\bigvee$-formula
Solution:

Davis, Logemann, Loveland 1962: "Splitting Rule":

1. Select an atom A
2. Split into cases A and $\neg\mathrm{A}$.
3. In each case, simplify according to new information.

Davis 1963; Chinlund, Davis, Hinman, McIlroy 1964:

Improvement of first-order case.

Overview

Propositional DPLL as a semantic tree method ✔

First-Order DPLL so far

FDPLL

Relation to other calculi

DPLL - The First-Order Case (1962)

Preprocessing:

Given Formula		Clause Form
$\forall x\ \exists y\ P(y,x)$ $\rightarrow \exists y\ P(y,a)$	$\longrightarrow$	$P(f(x),x)$ $\neg P(y,a)$

Outer loop:
Grounding

Inner loop:
Propositional
DPLL

DPLL - The First-Order Case (1962)

Preprocessing:

Given Formula

$$\forall x\ \exists y\ P(y,x) \rightarrow \exists y\ P(y,a)$$

Clause Form

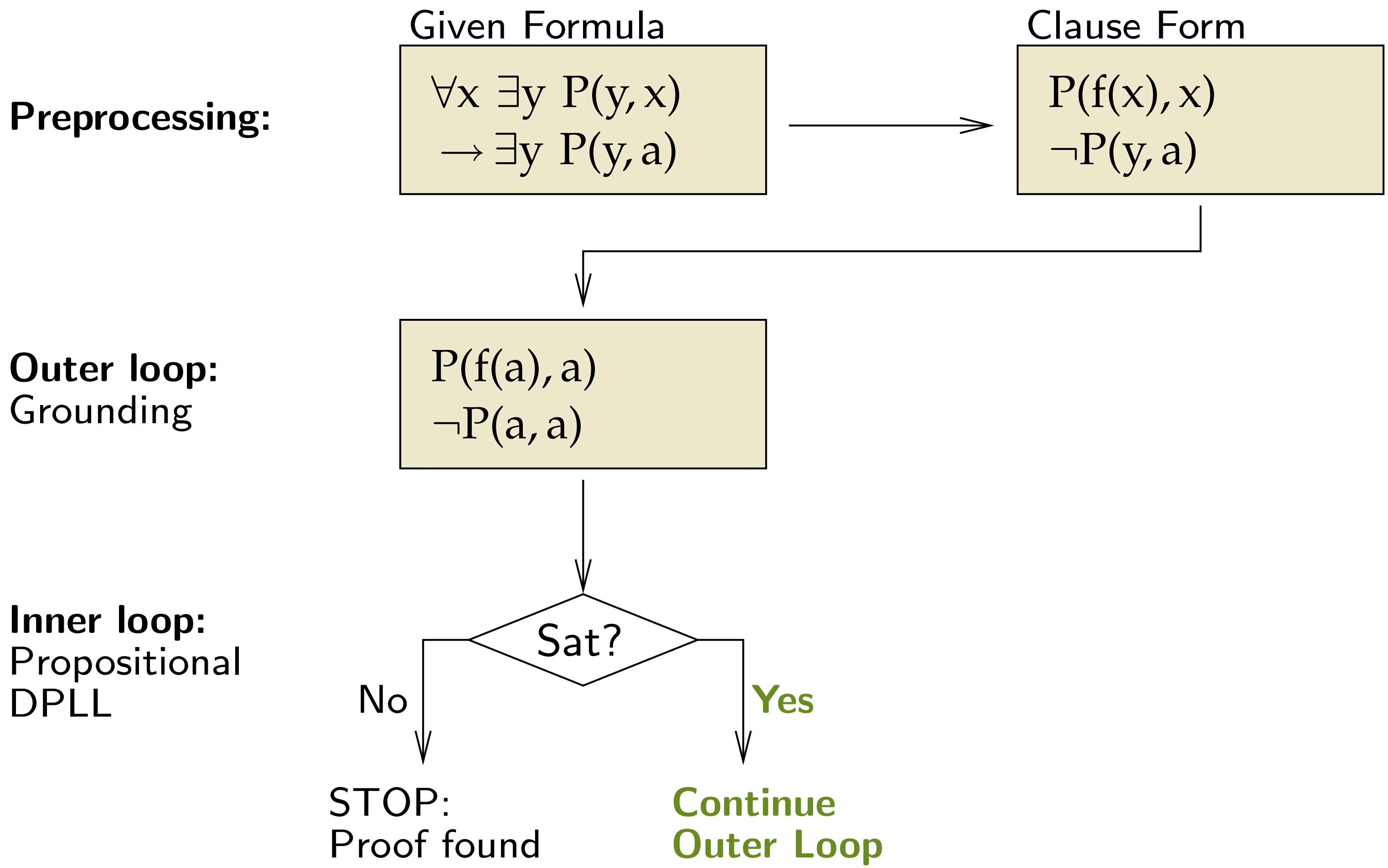
$$P(f(x),x)$$
$$\neg P(y,a)$$

Outer loop:
Grounding

$$P(f(a),a)$$
$$\neg P(a,a)$$

Inner loop:
Propositional
DPLL

DPLL - The First-Order Case (1962)



DPLL - The First-Order Case (1962)

Preprocessing:

Given Formula

$\forall x\ \exists y\ P(y,x)$
$\rightarrow \exists y\ P(y,a)$

Clause Form

$P(f(x),x)$
$\neg P(y,a)$

Outer loop:
Grounding

$P(f(a),a)$
$\neg P(a,a)$

$P(f(a),a)$
$\neg P(a,a)$
$\neg P(f(a),a)$

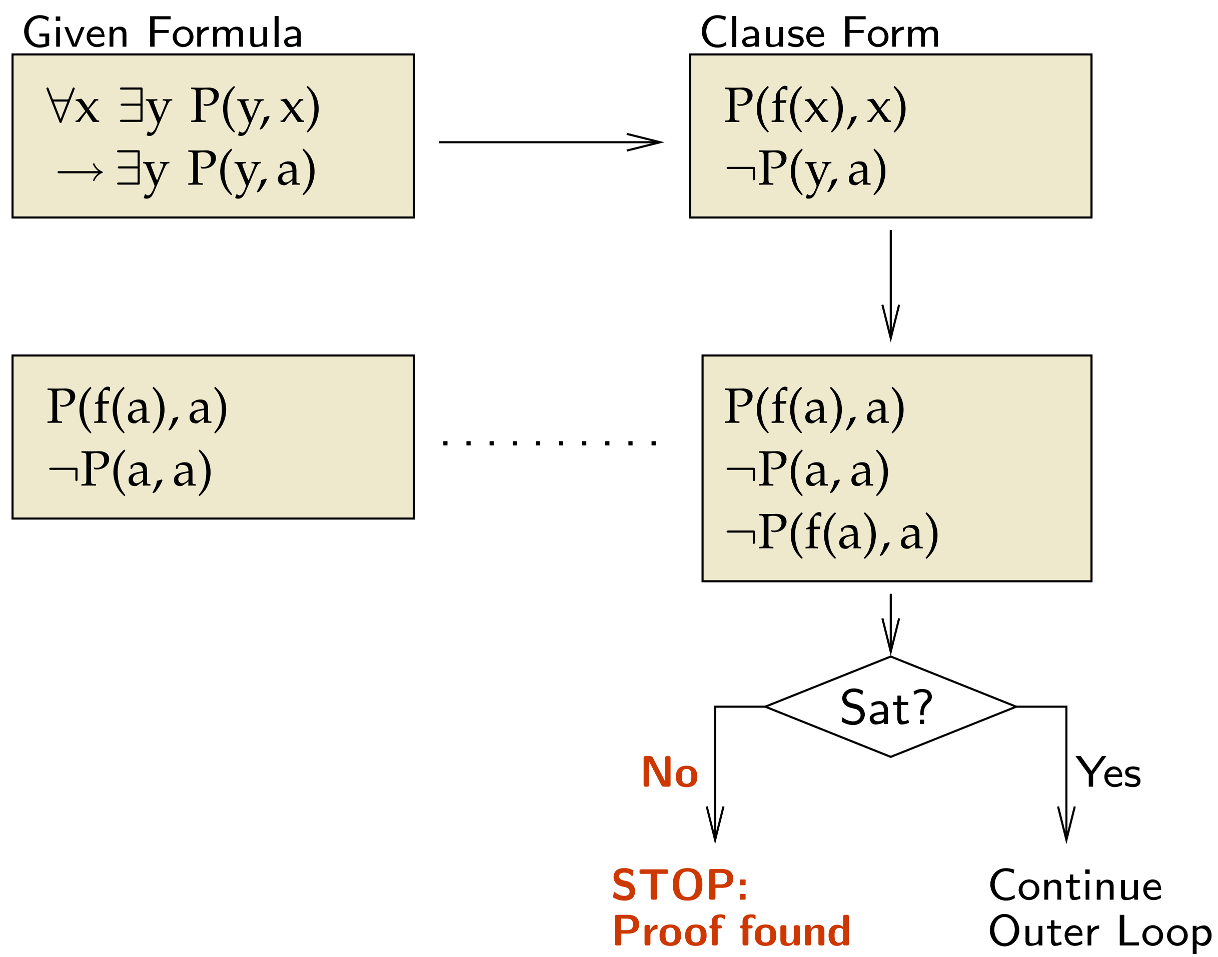
Inner loop:
Propositional
DPLL

DPLL - The First-Order Case (1962)

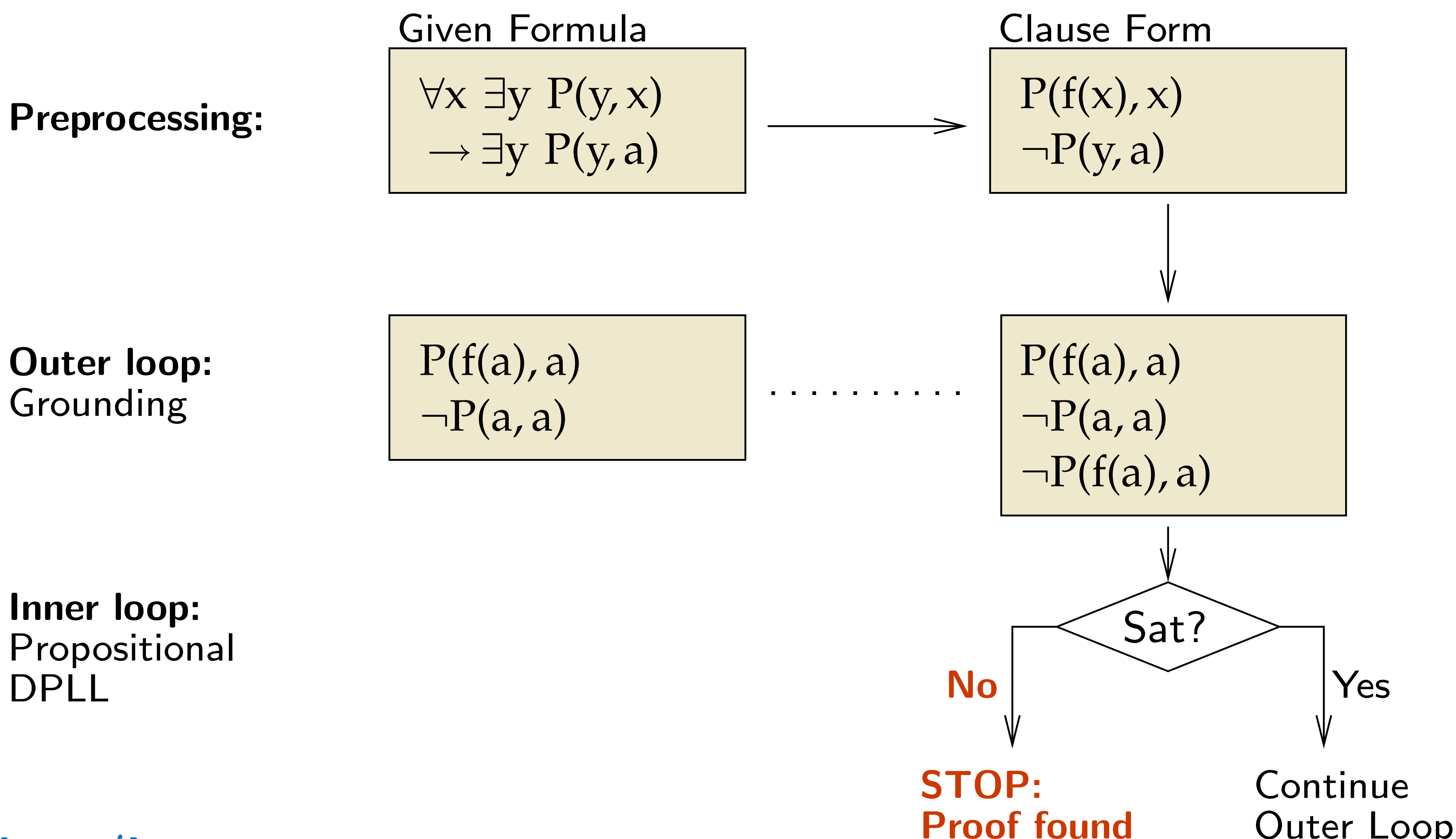
Preprocessing:

Outer loop:
Grounding

Inner loop:
Propositional
DPLL



DPLL - The First-Order Case (1962)



Problems/Issues:

- Controlling the grounding process in outer loop (irrelevant clauses)
- Repeat work across inner loops
- Weak redundancy criterion within inner loop

Controlling the Grounding Process

Davis 1963; Chinlund, Davis, Hinman, McIlroy 1964:

"Linked Conjunct Method":

Admissible clause set:	Non-admissible clause set:
$P(a) \lor Q(a)$	$P(b) \lor Q(a)$
$\neg P(a) \lor Q(a)$	$\neg P(a) \lor Q(a)$
$\neg Q(a) \lor P(a)$	$\neg Q(a) \lor P(a)$
Every literal has a mate	The literal $P(b)$ is pure

Anticipates unification! Note: Robinson paper on Resolution 1965

Controlling the Grounding Process

Davis 1963; Chinlund, Davis, Hinman, McIlroy 1964:

"Linked Conjunct Method":

Admissible clause set:	Non-admissible clause set:
$P(a) \vee Q(a)$	$P(b) \vee Q(a)$
$\neg P(a) \vee Q(a)$	$\neg P(a) \vee Q(a)$
$\neg Q(a) \vee P(a)$	$\neg Q(a) \vee P(a)$
Every literal has a mate	The literal $P(b)$ is pure

Anticipates unification! Note: Robinson paper on Resolution 1965

Some more recent work in this tradition:

Lee&Plaisted 1992, Chu&Plaisted 1994, Plaisted & Zhu 1997: (O)(S)HL

Billon 1996: Disconnection Method

Baumgartner 1998: Hyper Tableaux Next Generation

Parkes 1999: Lifted Search Engines for Satisfiability

May show very good performance!

Summary / Further Plan

Summary / Further Plan

- Instance based methods reduce first-order to propositional logic

Summary / Further Plan

- Instance based methods reduce first-order to propositional logic
- E.g. Resolution performs intrinsic first-order reasoning

  Advantages:

  Representation: Infinitely many inferences finitely represented:

$$\frac{P(f(x),x) \qquad \neg P(y,z) \lor Q(y,z)}{Q(f(x),x)}$$

Infinitely many inferences in instance based methods

Summary / Further Plan

- Instance based methods reduce first-order to propositional logic

- E.g. Resolution performs intrinsic first-order reasoning

  Advantages:

  Representation: Infinitely many inferences finitely represented:

$$\frac{P(f(x),x) \qquad \neg P(y,z) \lor Q(y,z)}{Q(f(x),x)}$$

  Infinitely many inferences in instance based methods

  Redundancy testing: E.g. by subsumption:

$$\neg P(y,z) \quad \text{subsumes} \quad \neg P(y,y) \lor Q(y,y)$$

  Lack of redundancy testing in instance based methods

Overview

Propositional DPLL as a semantic tree method ✔

First-Order DPLL so far ✔

FDPLL

Relation to other calculi

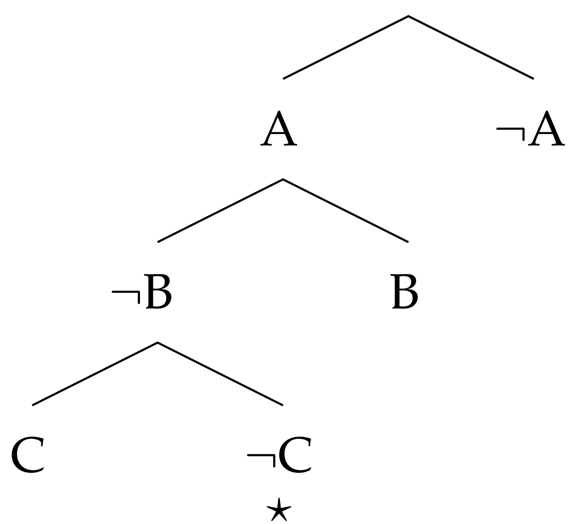
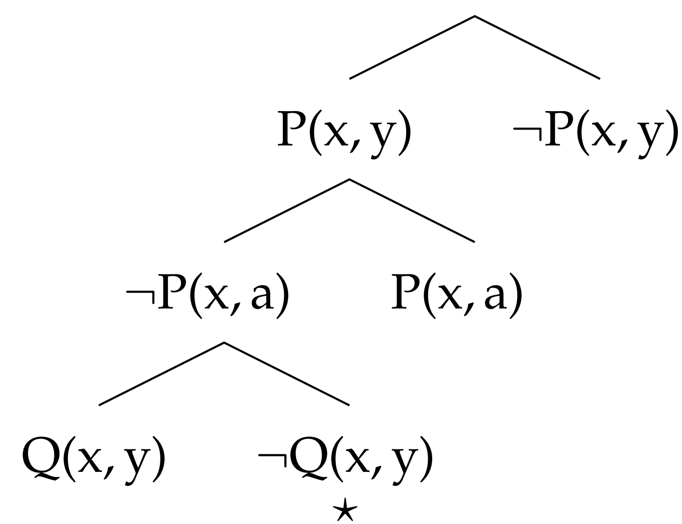
Meta-Level Strategy

Lifted data structures:

	Propositional Reasoning	First-Order Reasoning
Resolution	$A \vee \neg B \vee C$	$P(x, y) \vee \neg Q(x, z) \vee R(y, z)$

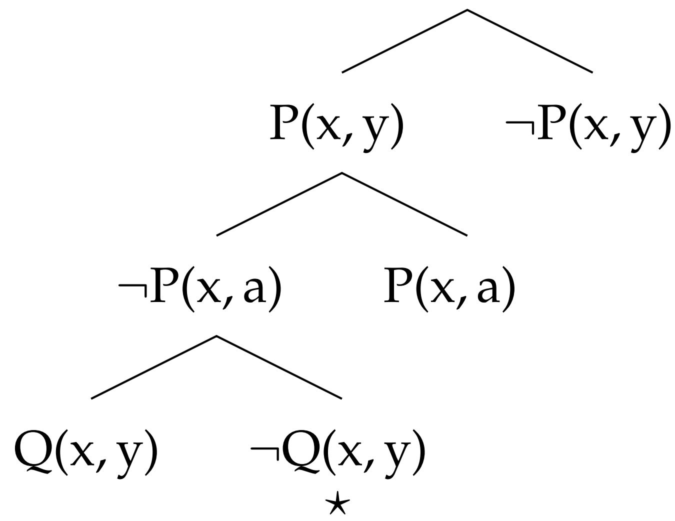
Meta-Level Strategy

Lifted data structures:

	Propositional Reasoning	First-Order Reasoning
Resolution	$A \vee \neg B \vee C$	$P(x, y) \vee \neg Q(x, z) \vee R(y, z)$
DPLL		

FDPLL: First-Order Semantic Trees

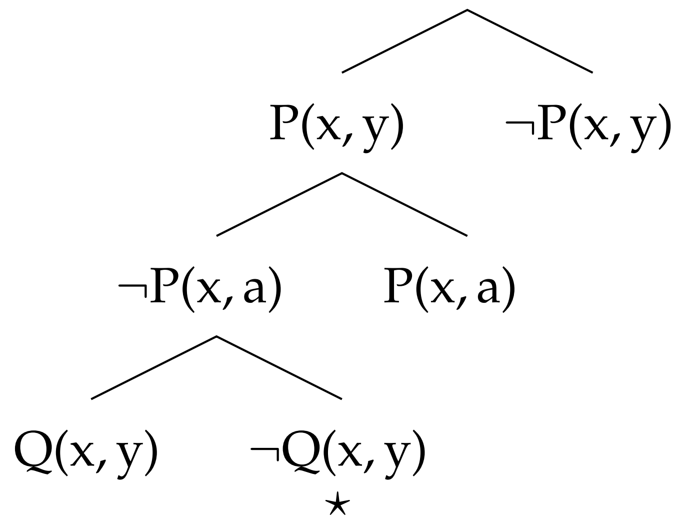
First-Order Semantic Trees



Issues:

- One-branch-at-a-time approach desired

First-Order Semantic Trees

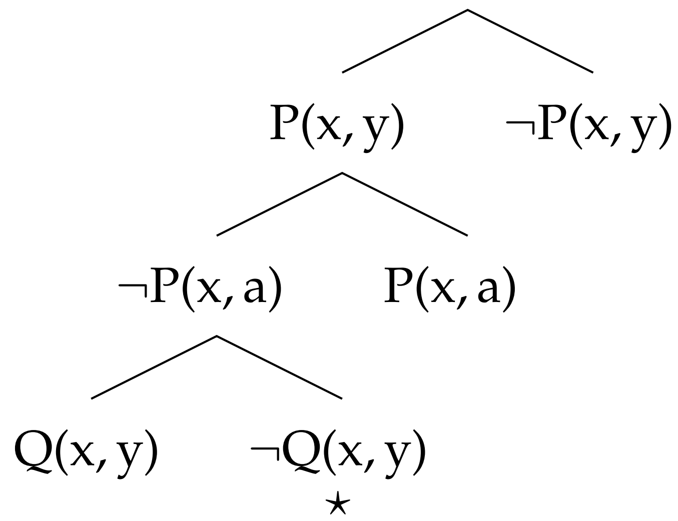


Issues:

- One-branch-at-a-time approach desired
- How are variables treated?

  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux? (c) Schema!

First-Order Semantic Trees

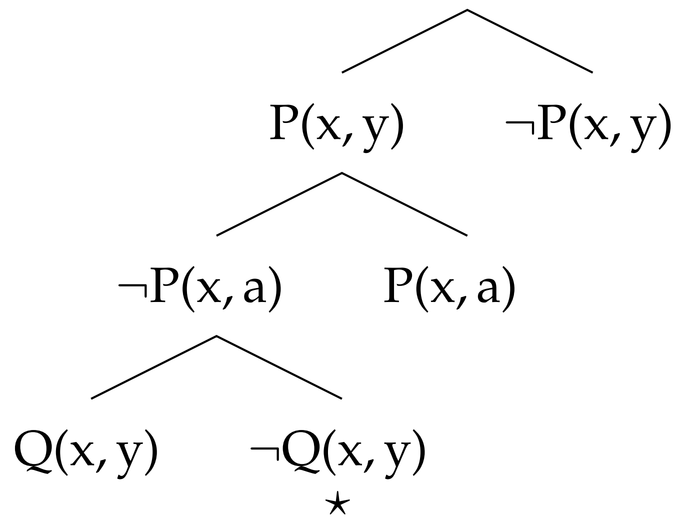


Issues:

- One-branch-at-a-time approach desired
- How are variables treated?

  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux? (c) Schema!
- How to extract an interpretation from a branch?

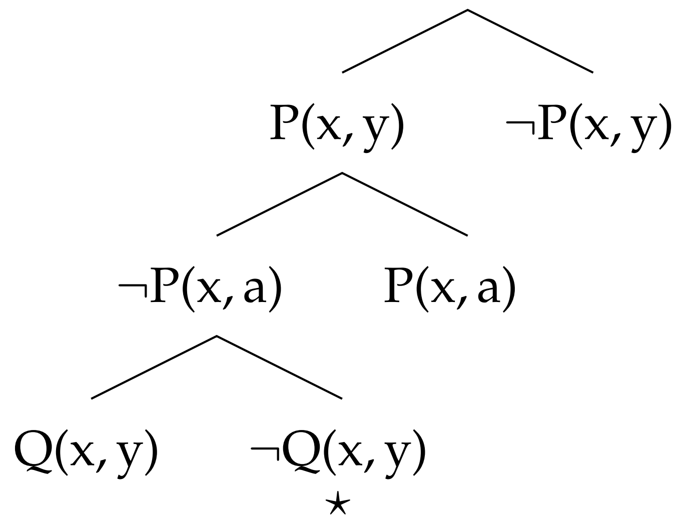
First-Order Semantic Trees



Issues:

- One-branch-at-a-time approach desired
- How are variables treated?
  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux? (c) Schema!
- How to extract an interpretation from a branch?
- When is a branch closed?

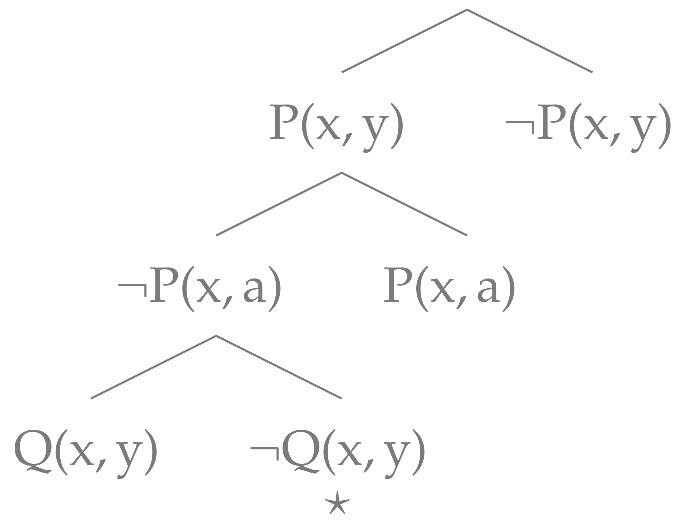
First-Order Semantic Trees



Issues:

- One-branch-at-a-time approach desired
- How are variables treated?
  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux? (c) Schema!
- How to extract an interpretation from a branch?
- When is a branch closed?
- How to construct such trees (calculus)?

First-Order Semantic Trees



Issues:

- One-branch-at-a-time approach desired
- How are variables treated?
  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux? (c) Schema!
- **How to extract an interpretation from a branch?**
- When is a branch closed?
- How to construct such trees (calculus)?

Extracting an Interpretation from a Branch

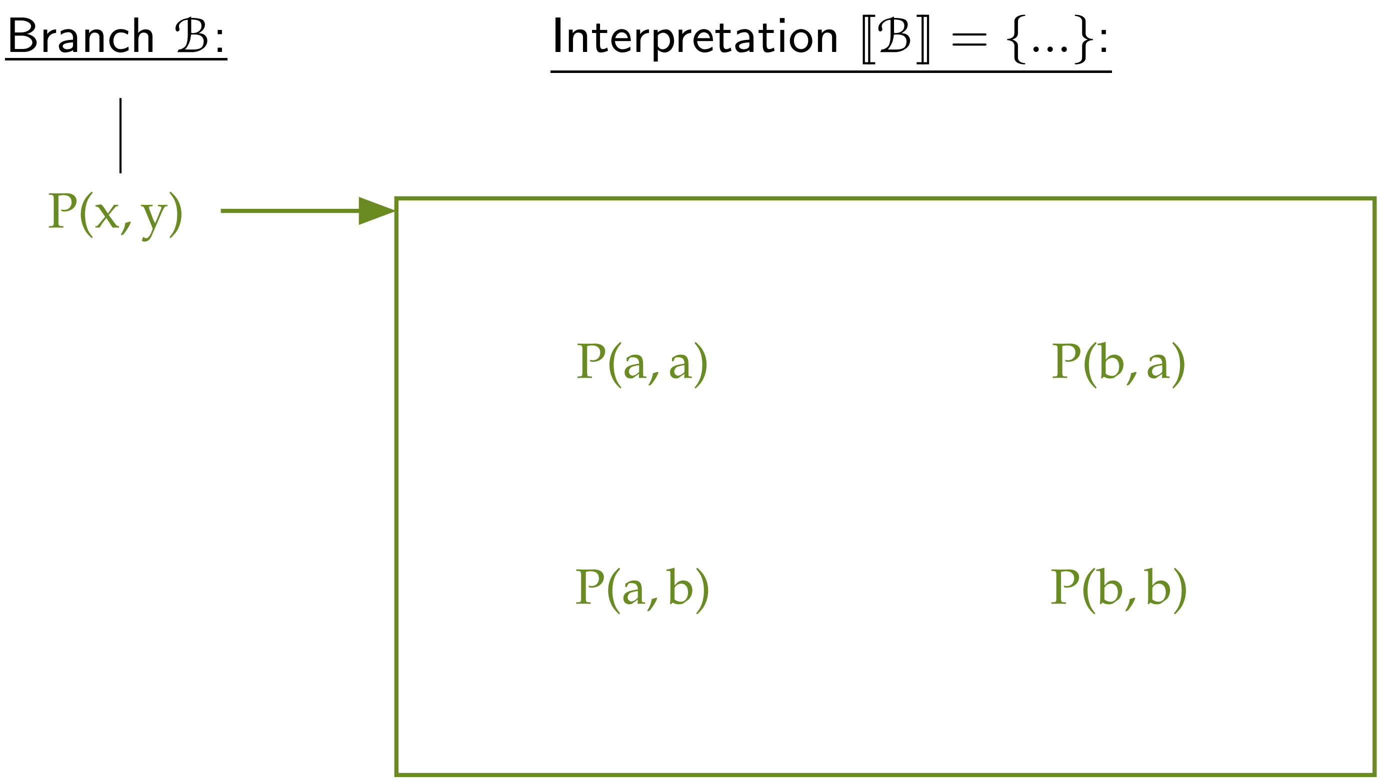
Branch $\mathcal{B}$:

$$\Big| \\ P(x, y)$$

Interpretation $[\![\mathcal{B}]\!] = \{\ldots\}$:

- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values.

Extracting an Interpretation from a Branch



- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values.

Extracting an Interpretation from a Branch

Branch $\mathcal{B}$:

$P(x, y)$

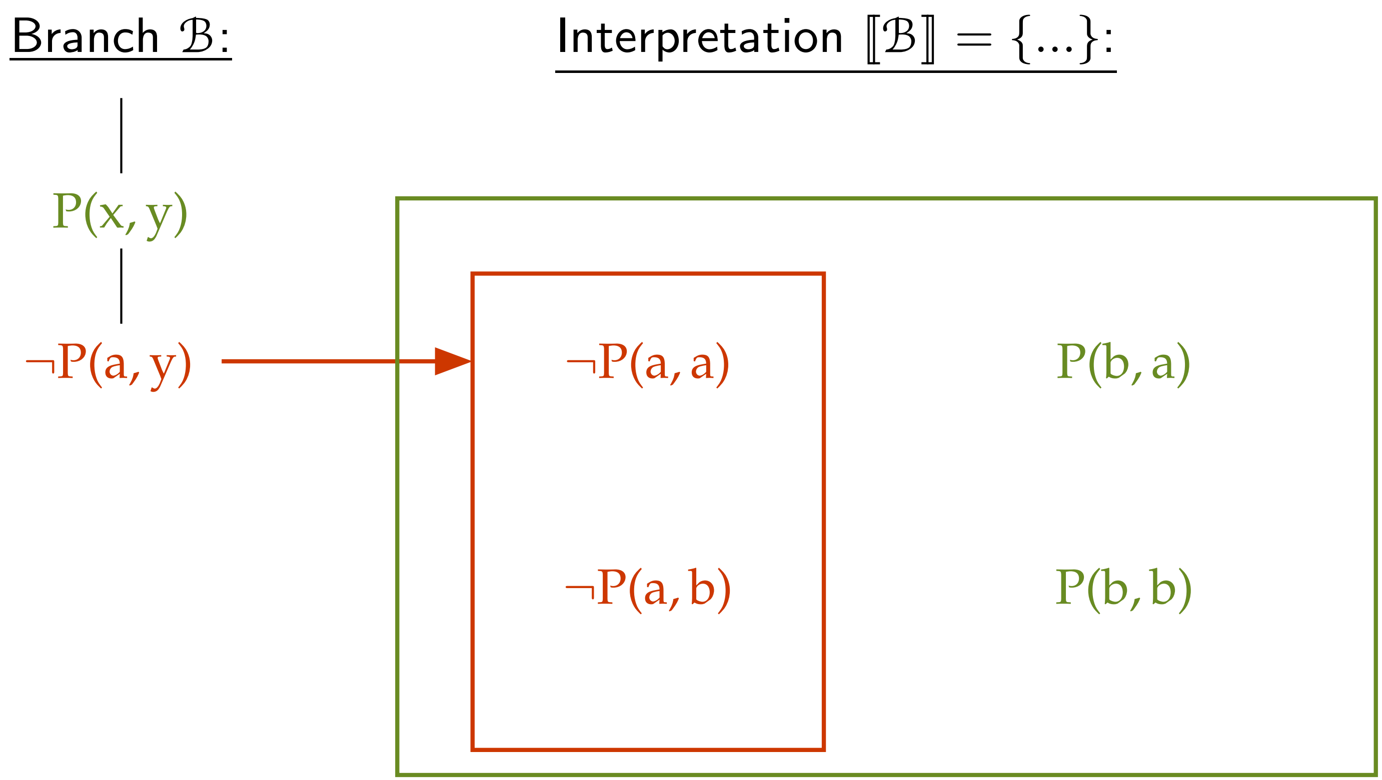
$\neg P(a, y)$

Interpretation $[\![\mathcal{B}]\!] = \{\ldots\}$:

$P(a, a)$	$P(b, a)$
$P(a, b)$	$P(b, b)$

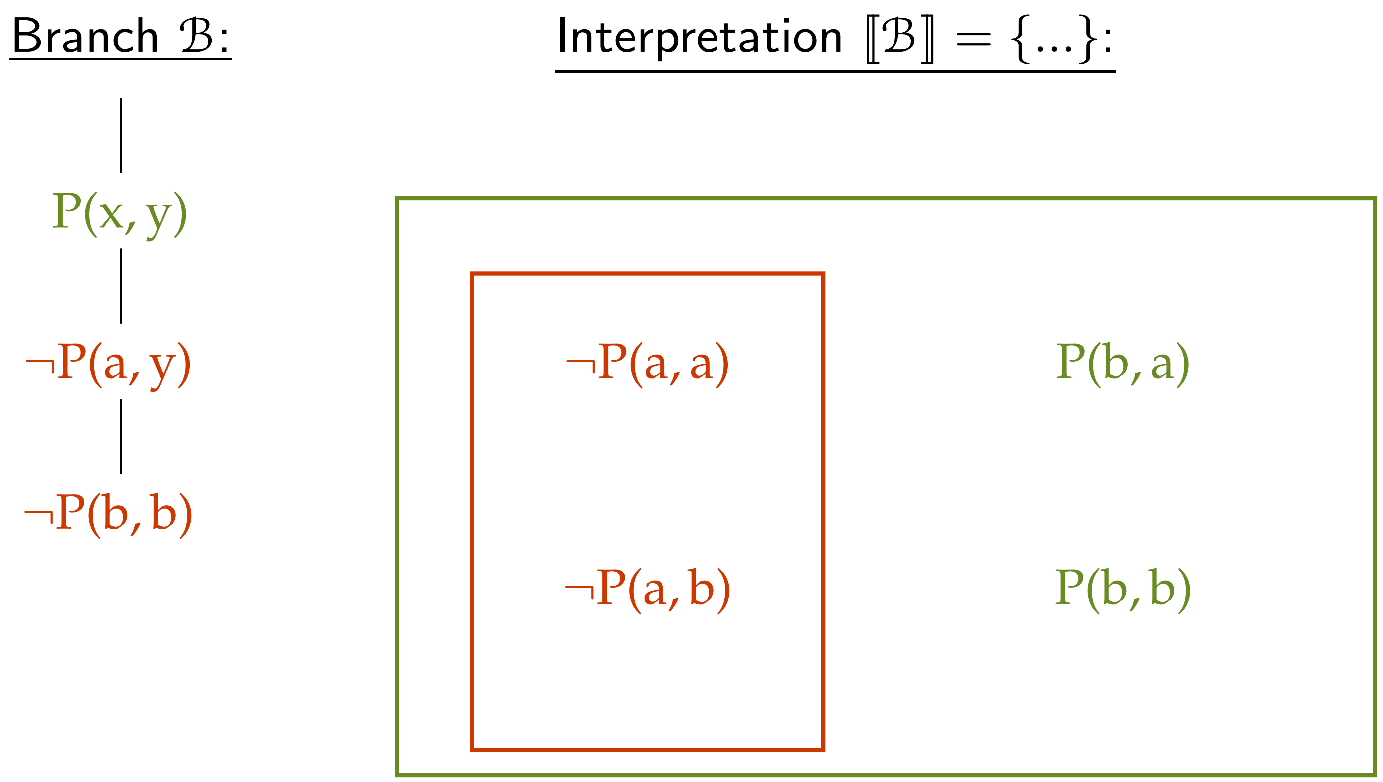
- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values.

Extracting an Interpretation from a Branch



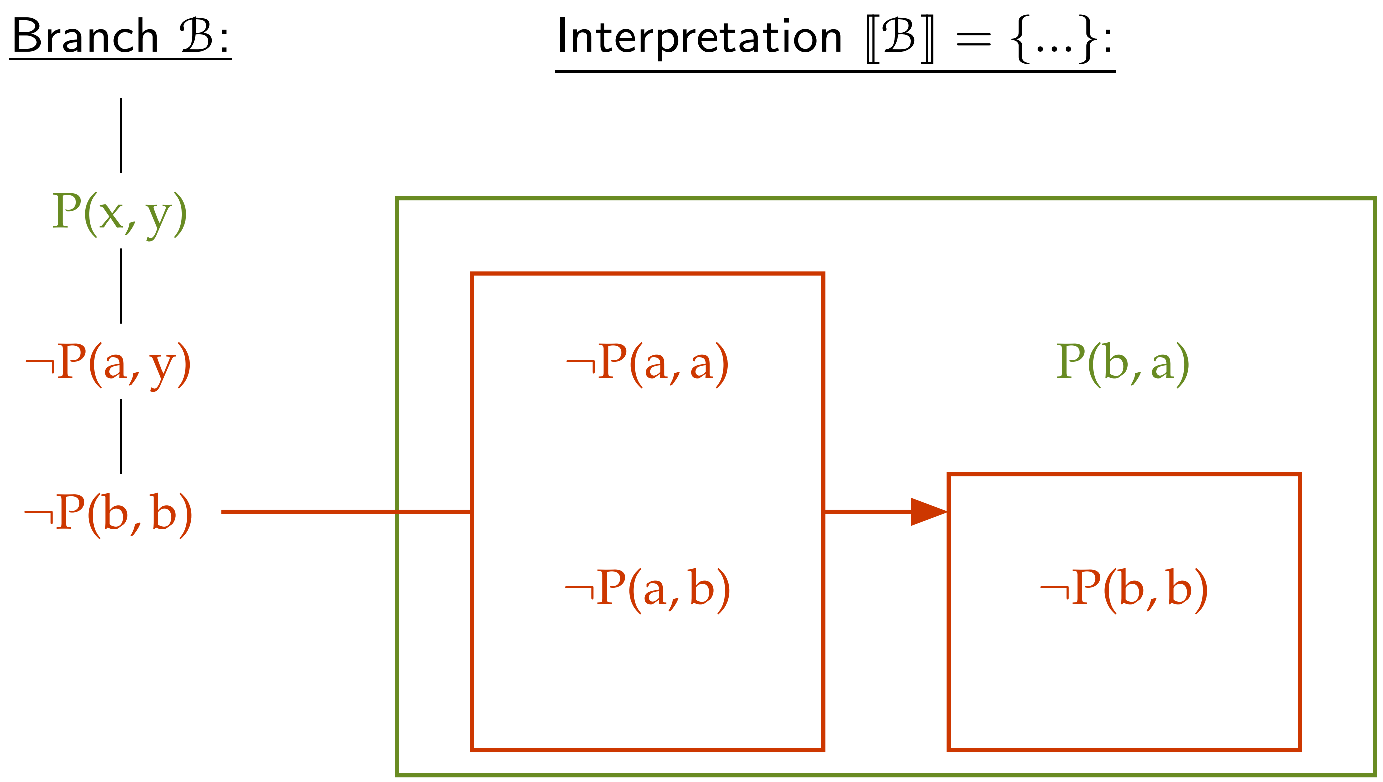
- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values.

Extracting an Interpretation from a Branch



- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values.

Extracting an Interpretation from a Branch

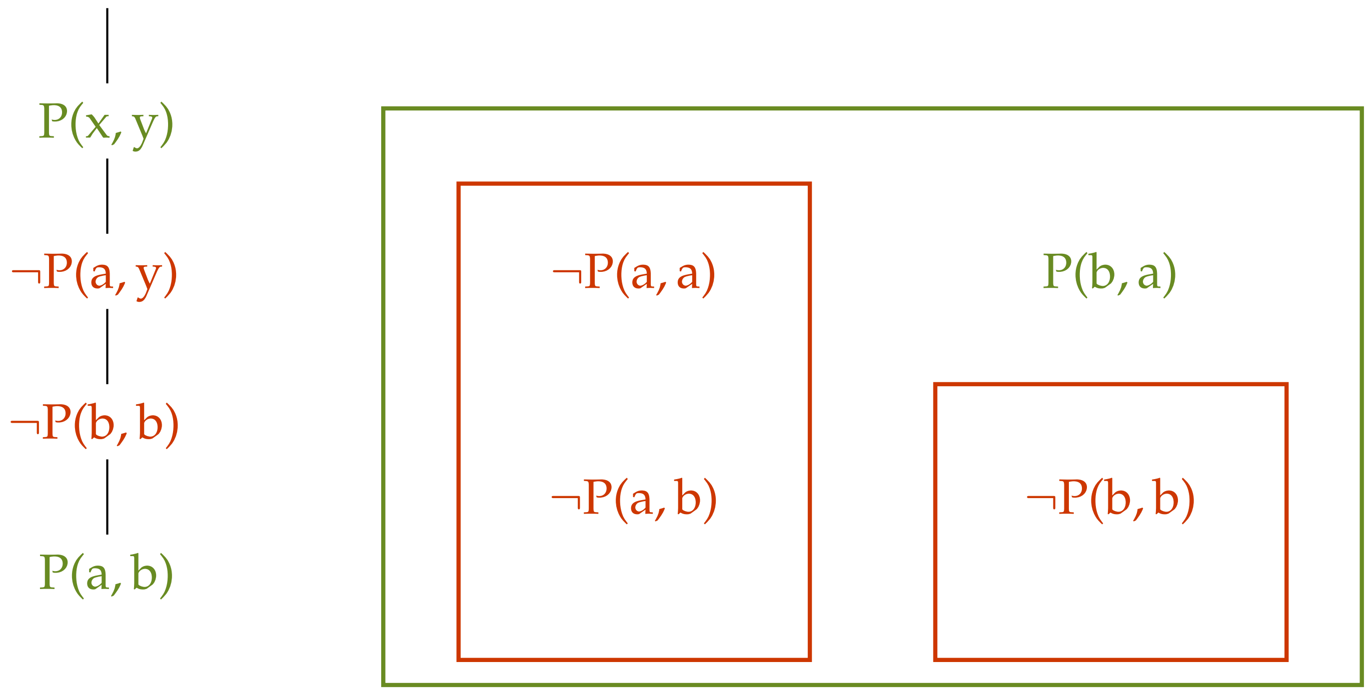


- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values.

Extracting an Interpretation from a Branch

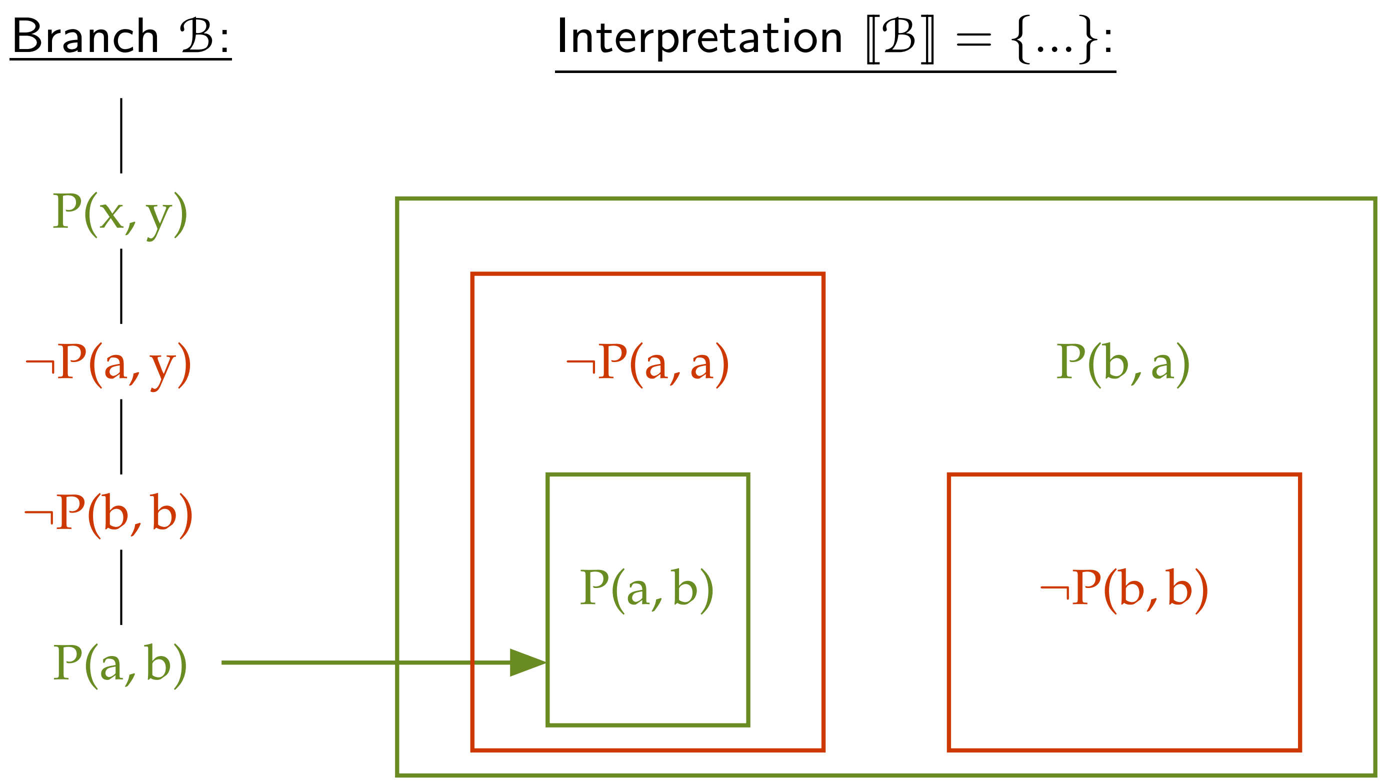
Branch $\mathcal{B}$:

Interpretation $[\![\mathcal{B}]\!] = \{\ldots\}$:



- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values.

Extracting an Interpretation from a Branch



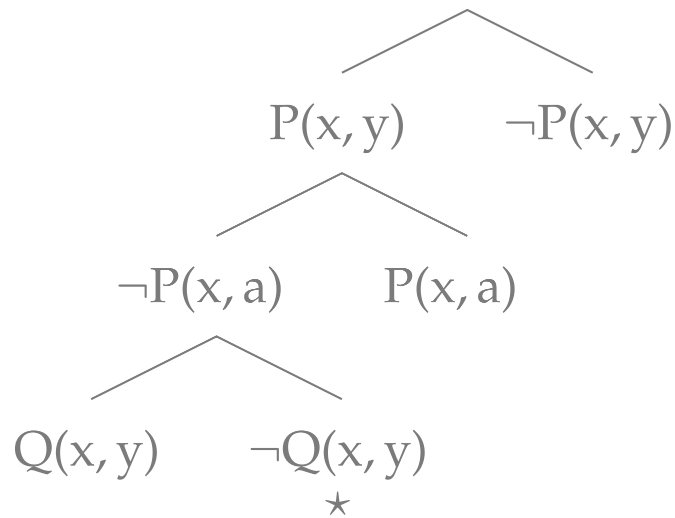
- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values.

Extracting an Interpretation from a Branch

Branch $\mathcal{B}$:	Interpretation $[\![\mathcal{B}]\!] = \{\ldots\}$:
$P(x,y)$	
$\neg P(a,y)$	$\{\ \neg P(a,a),\quad P(b,a),$
$\neg P(b,b)$	
	$P(a,b),\quad \neg P(b,b)\ \}$
$P(a,b)$	

- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying opposite truth values.
- The order of literals does not matter.

First-Order Semantic Trees



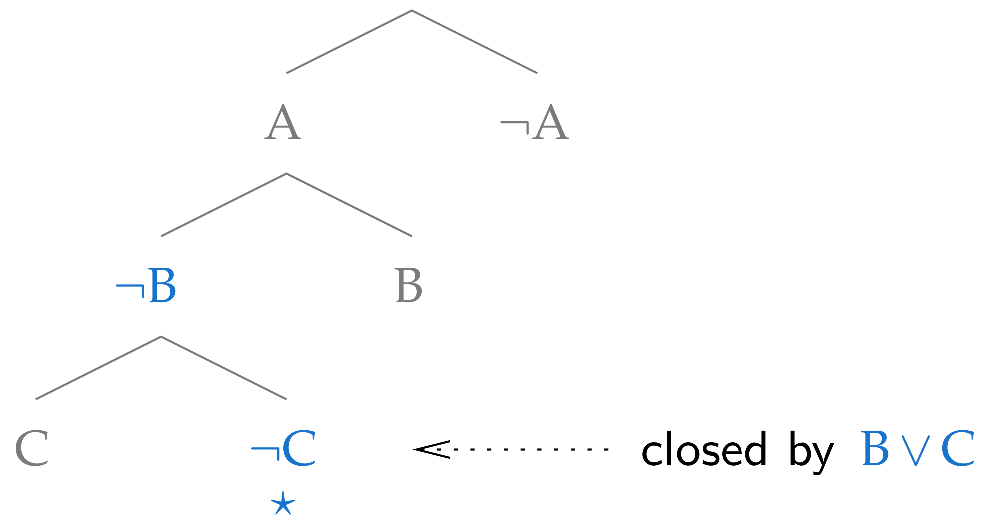
Issues:

- One-branch-at-a-time approach desired
- How are variables treated?
  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux? (c) Schema!
- How to extract an interpretation from a branch? ✔
- **When is a branch closed?**
- How to construct such trees (calculus)?

Calculus: Branch Closure

Purpose: Determine if branch elementary contradicts an input clause.

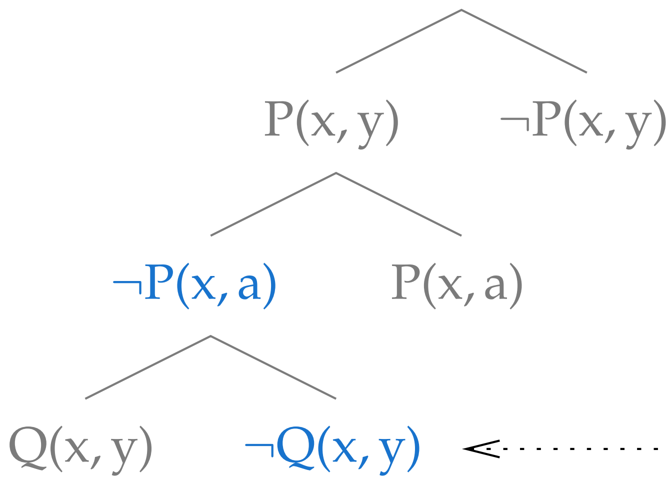
Propositional case:



Calculus: Branch Closure

Purpose: Determine if branch elementary contradicts an input clause.

FDPLL case:

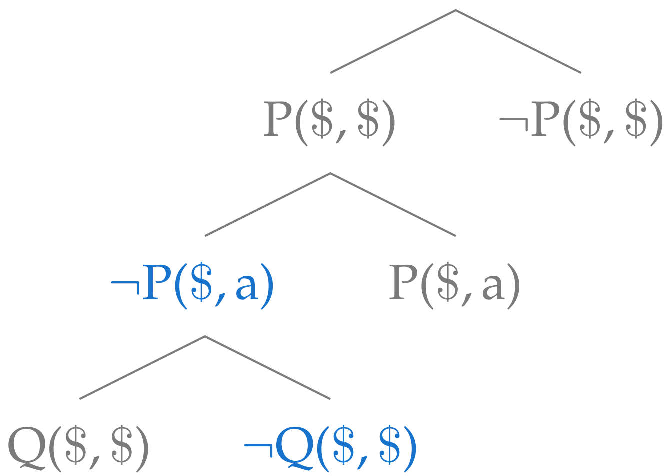


closed by $P(x,y) \vee Q(x,x)$?

Calculus: Branch Closure

Purpose: Determine if branch elementary contradicts an input clause.

FDPLL case:



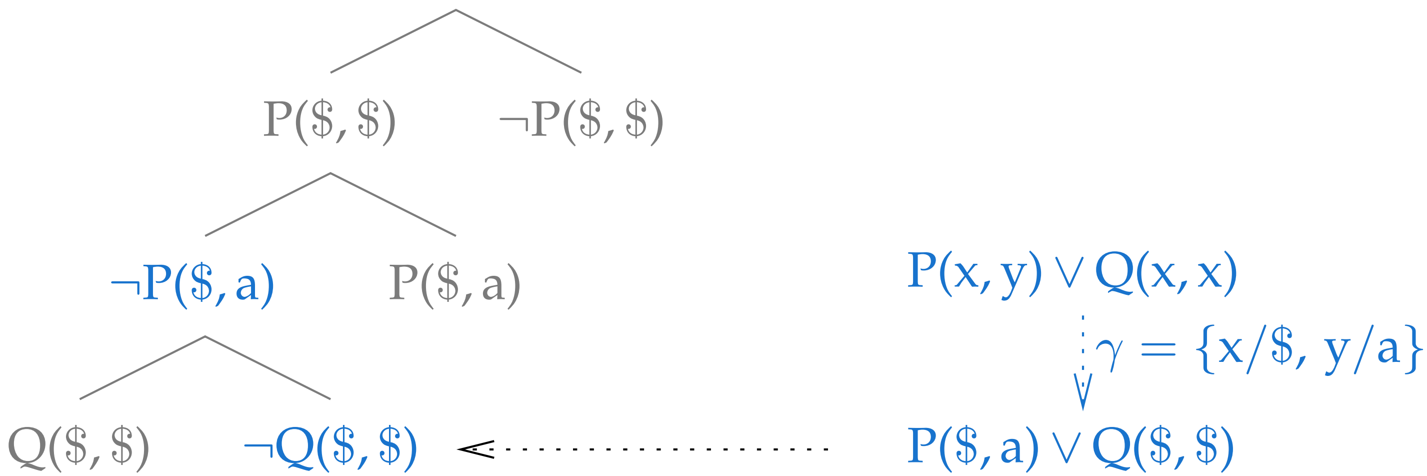
$P(x, y) \lor Q(x, x)$

1. Replace all variables in tree by a constant \$. Gives propositional tree
2.
3.

Calculus: Branch Closure

Purpose: Determine if branch elementary contradicts an input clause.

FDPLL case:

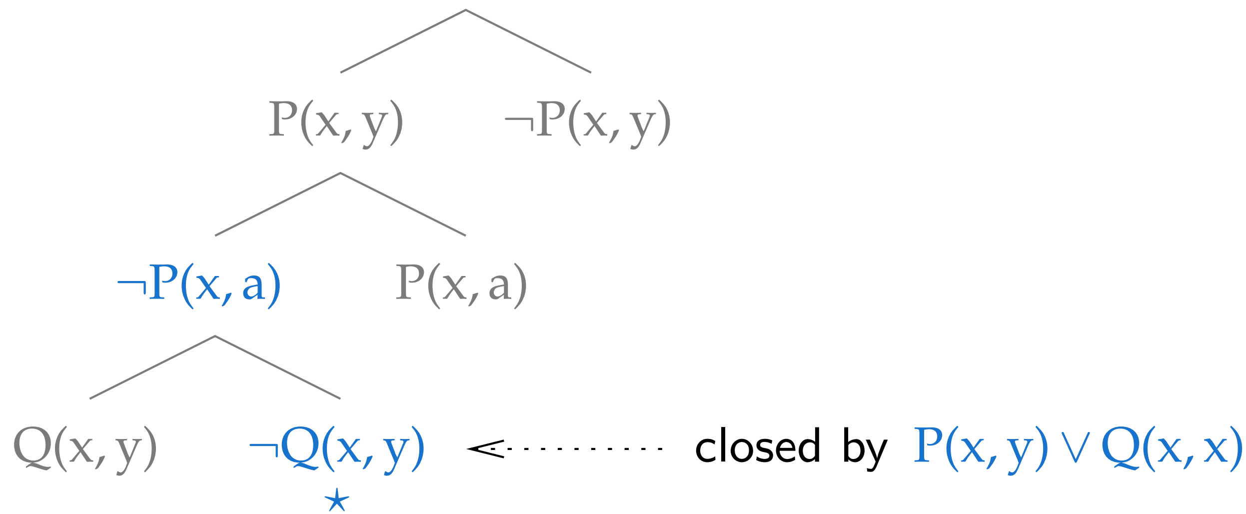


1. Replace all variables in tree by a constant $\$$. Gives propositional tree
2. Compute matcher γ to propositionally close branch
3.

Calculus: Branch Closure

Purpose: Determine if branch elementary contradicts an input clause.

FDPLL case:

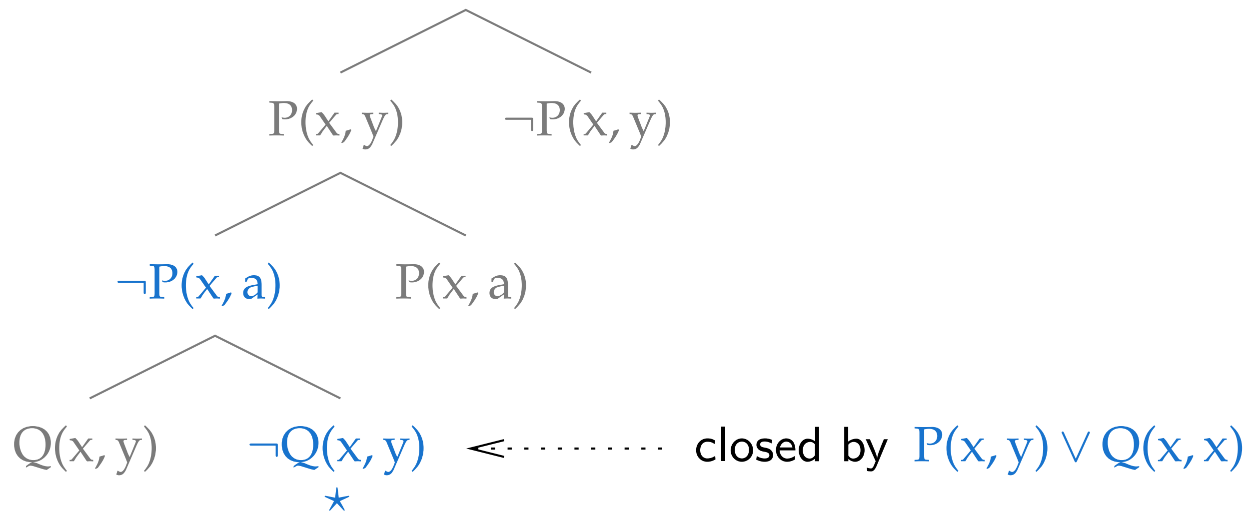


1. Replace all variables in tree by a constant $\$$. Gives propositional tree
2. Compute matcher γ to propositionally close branch
3. Mark branch as closed $(\star)$

Calculus: Branch Closure

Purpose: Determine if branch elementary contradicts an input clause.

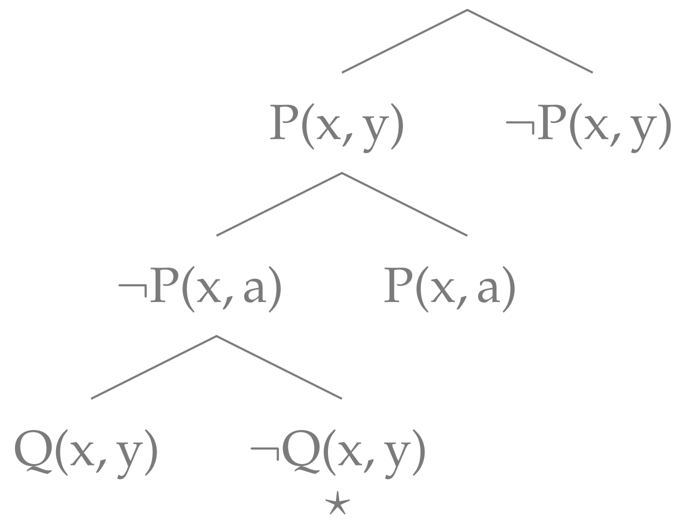
FDPLL case:



1. Replace all variables in tree by a constant $\$$. Gives propositional tree
2. Compute matcher γ to propositionally close branch
3. Mark branch as closed ($\star$)

Theorem: FDPLL is sound (because propositional DPLL is sound), and splitting can be done with arbitrary literal.

First-Order Semantic Trees



Issues:

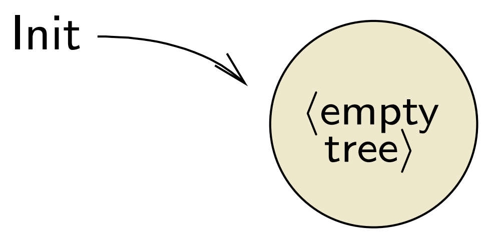
- How are variables treated?
  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux? (c) Schema!
- How to extract an interpretation from a branch? ✔
- When is a branch closed? ✔
- **How to construct such trees (calculus)?**

FDPLL Calculus

Input: a clause set $\mathcal{S}$

Output: "unsatisfiable" or "satisfiable" (if terminates)

Note: Strategy much like in inner loop of propositional DPLL:

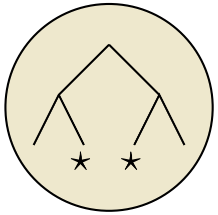


FDPLL Calculus

Input: a clause set $\mathcal{S}$

Output: “unsatisfiable” or “satisfiable” (if terminates)

Note: Strategy much like in inner loop of propositional DPLL:

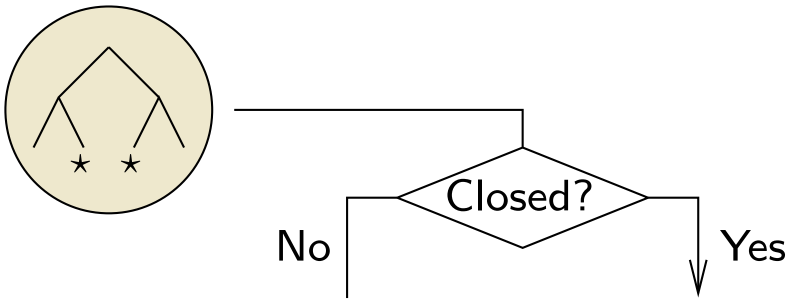


FDPLL Calculus

Input: a clause set $\mathcal{S}$

Output: "unsatisfiable" or "satisfiable" (if terminates)

Note: Strategy much like in inner loop of propositional DPLL:

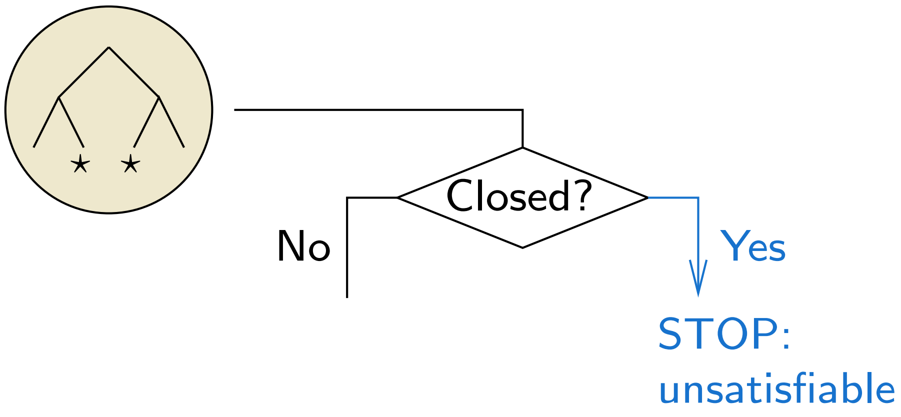


FDPLL Calculus

Input: a clause set $\mathcal{S}$

Output: "unsatisfiable" or "satisfiable" (if terminates)

Note: Strategy much like in inner loop of propositional DPLL:

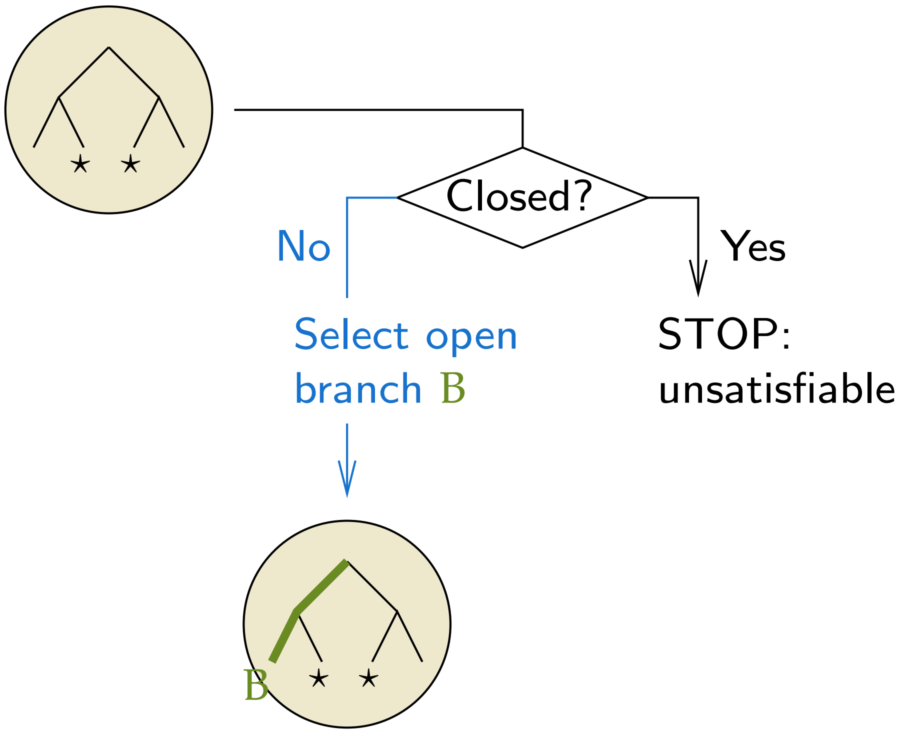


FDPLL Calculus

Input: a clause set $\mathcal{S}$

Output: "unsatisfiable" or "satisfiable" (if terminates)

Note: Strategy much like in inner loop of propositional DPLL:

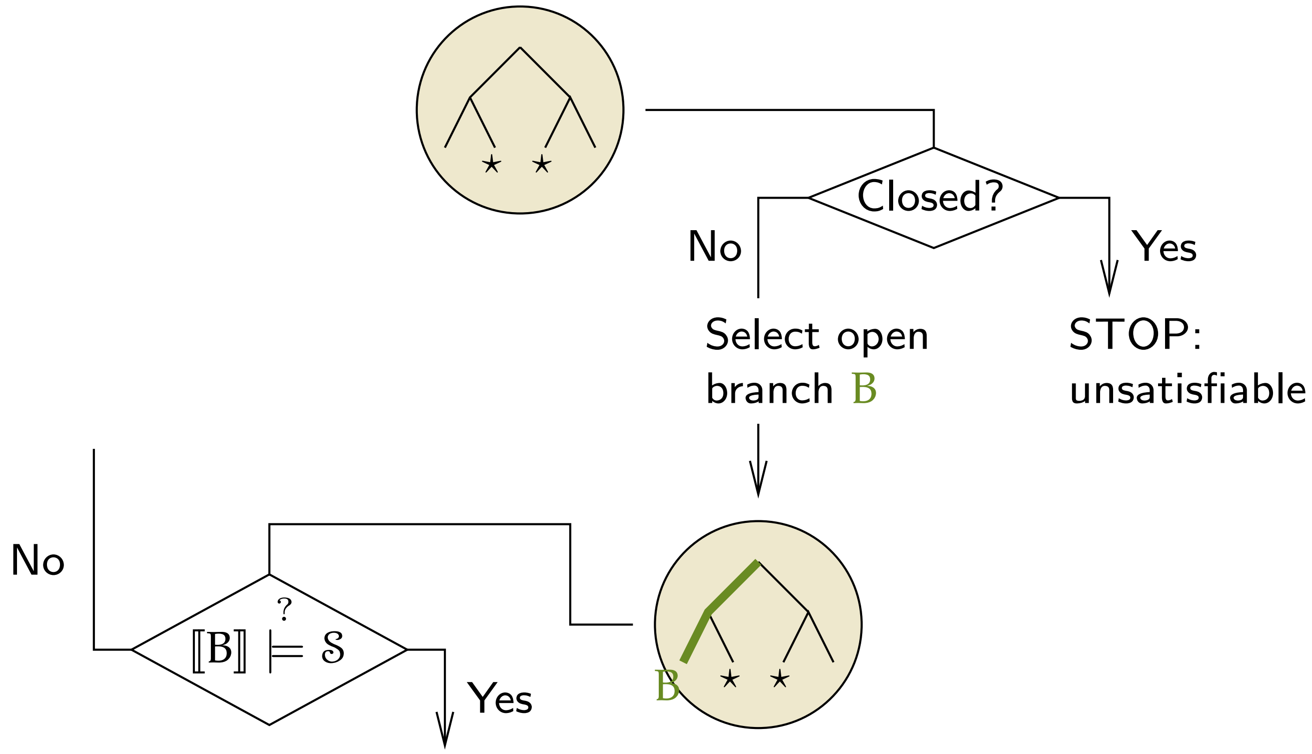


FDPLL Calculus

Input: a clause set $\mathcal{S}$

Output: “unsatisfiable” or “satisfiable” (if terminates)

Note: Strategy much like in inner loop of propositional DPLL:

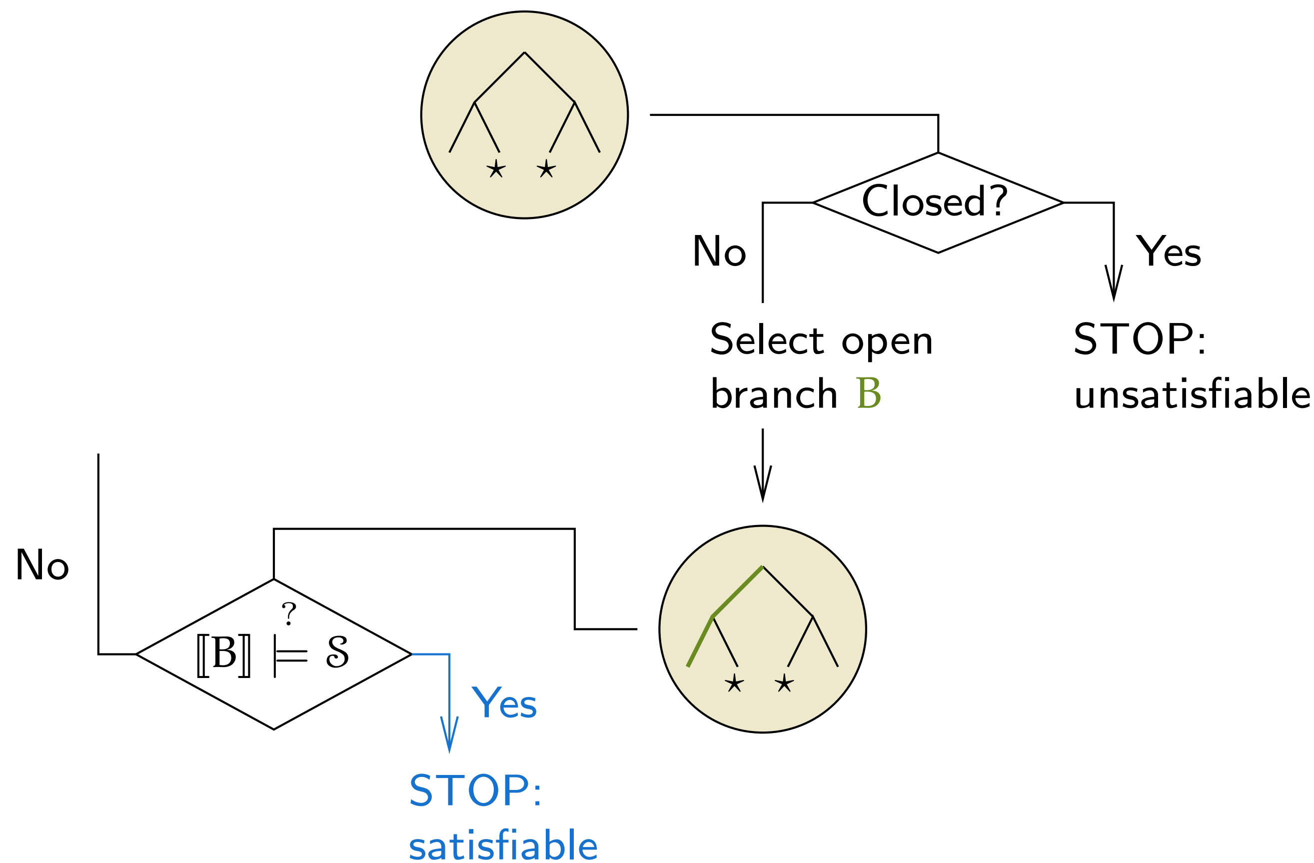


FDPLL Calculus

Input: a clause set $\mathcal{S}$

Output: "unsatisfiable" or "satisfiable" (if terminates)

Note: Strategy much like in inner loop of propositional DPLL:

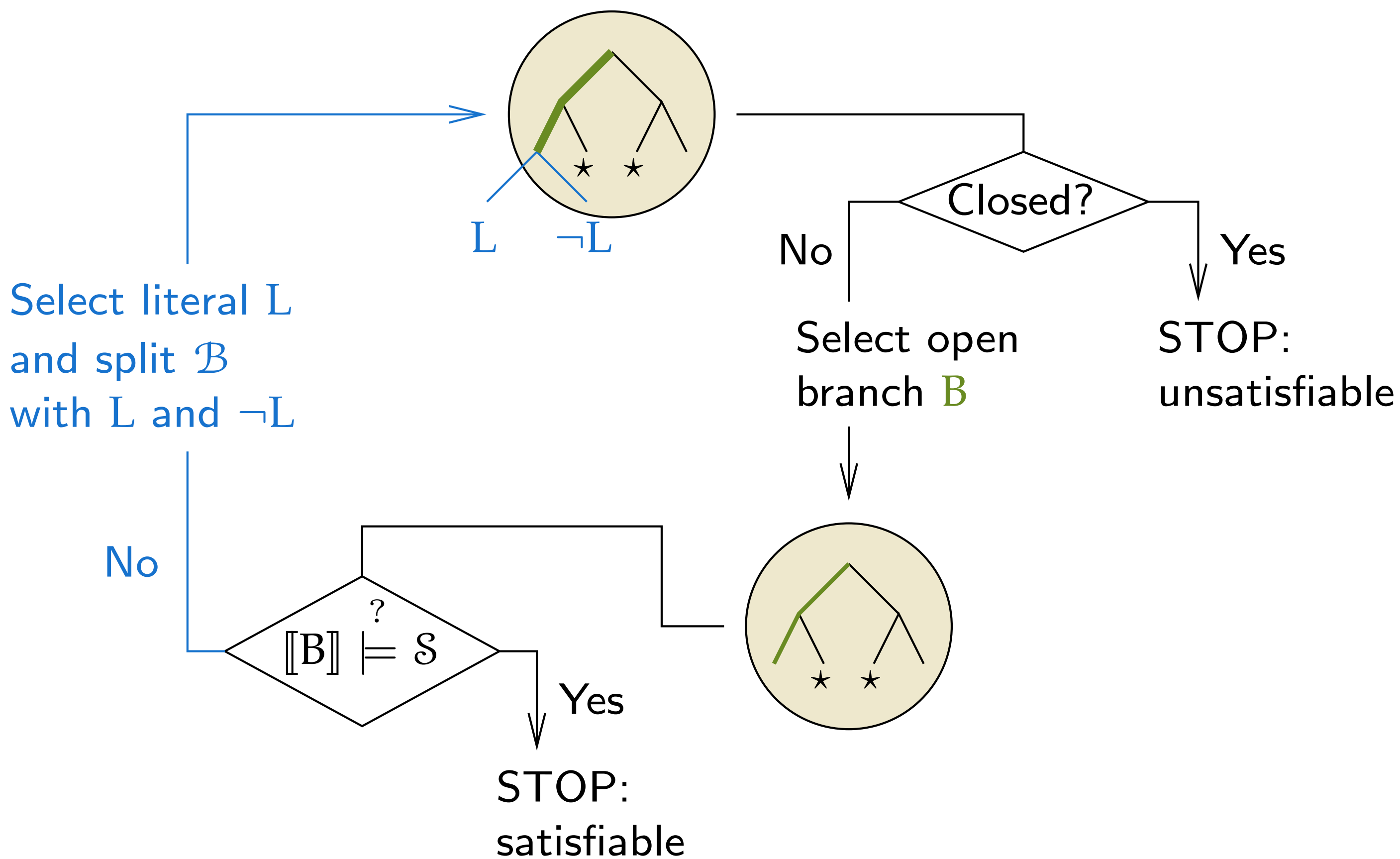


FDPLL Calculus

Input: a clause set $\mathcal{S}$

Output: "unsatisfiable" or "satisfiable" (if terminates)

Note: Strategy much like in inner loop of propositional DPLL:

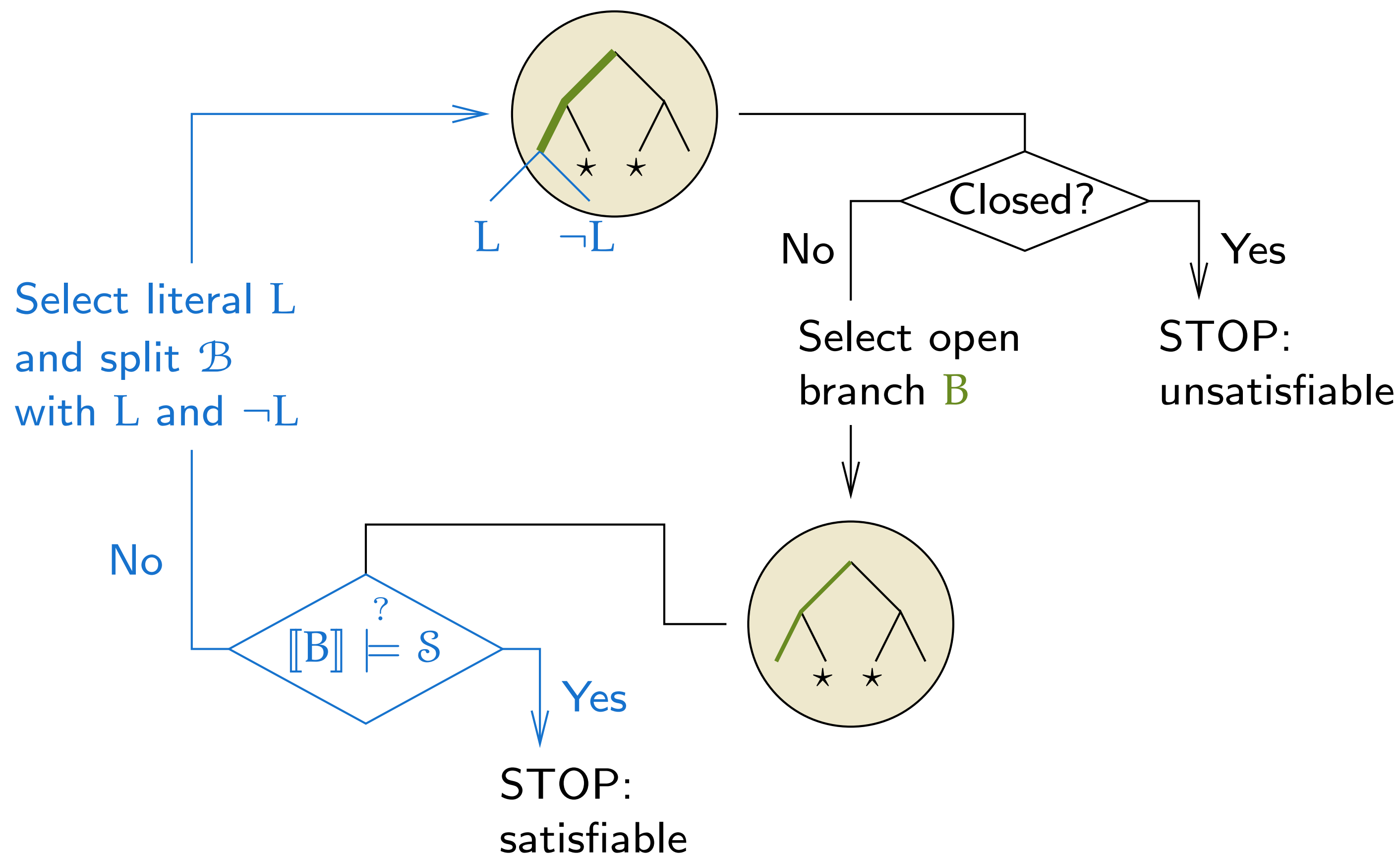


FDPLL Calculus

Input: a clause set $\mathcal{S}$

Output: "unsatisfiable" or "satisfiable" (if terminates)

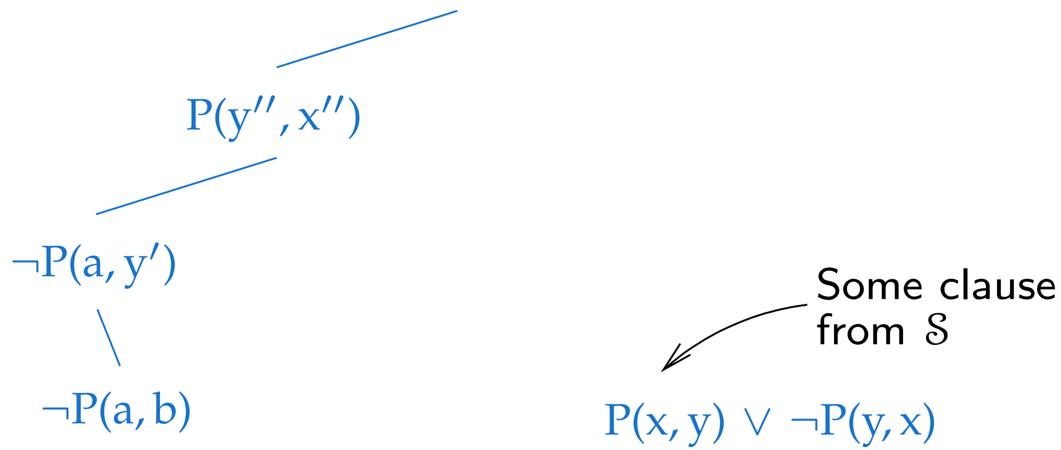
Note: Strategy much like in inner loop of propositional DPLL:



Next: Testing $[\![B]\!] \models \mathcal{S}$ and splitting

Calculus: The Splitting Rule

Purpose: Satisfy a clause that is currently "false"



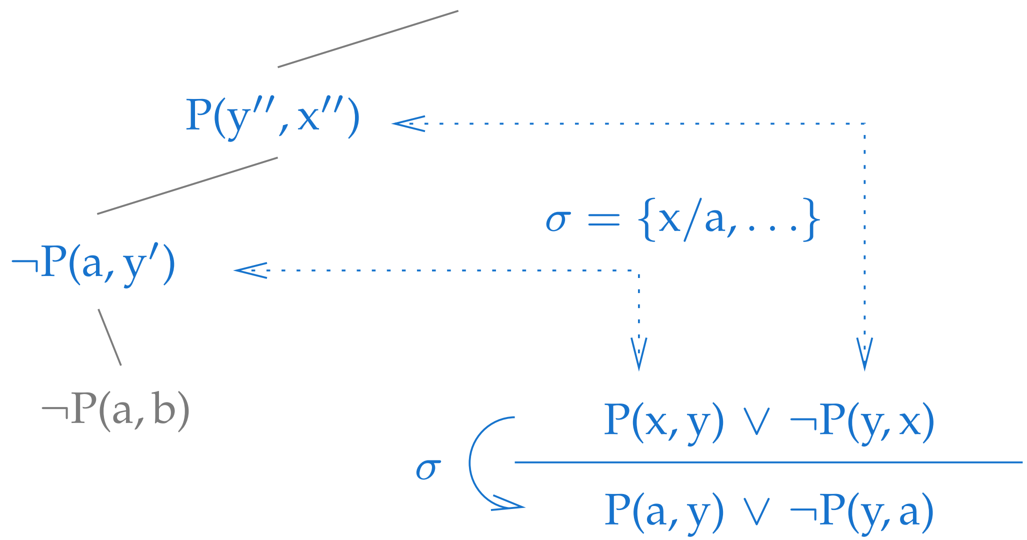
1.

2.

3.

Calculus: The Splitting Rule

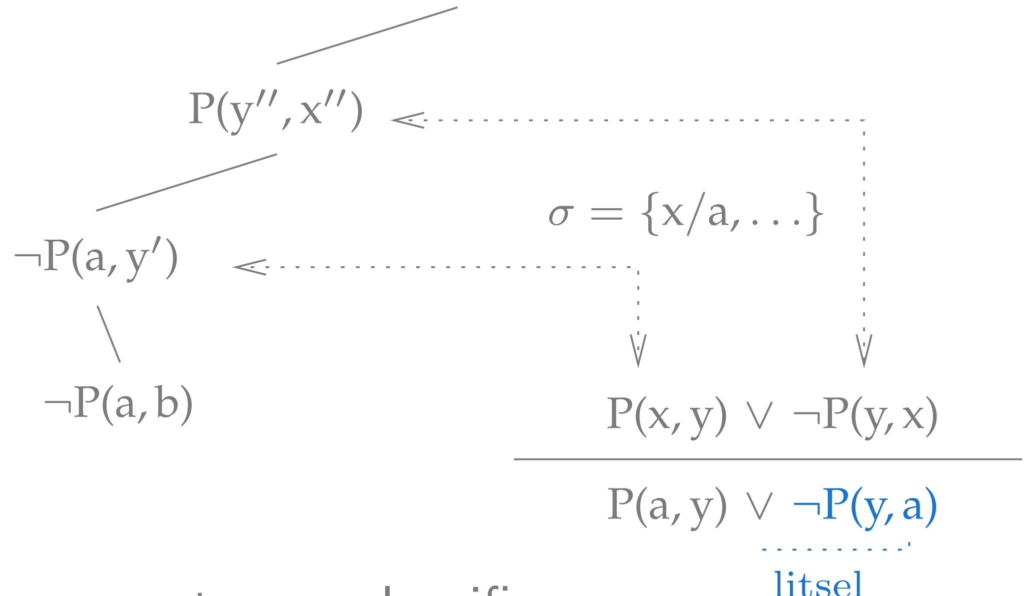
Purpose: Satisfy a clause that is currently “false”



1. Compute simultaneous most general unifier σ
2.
3.

Calculus: The Splitting Rule

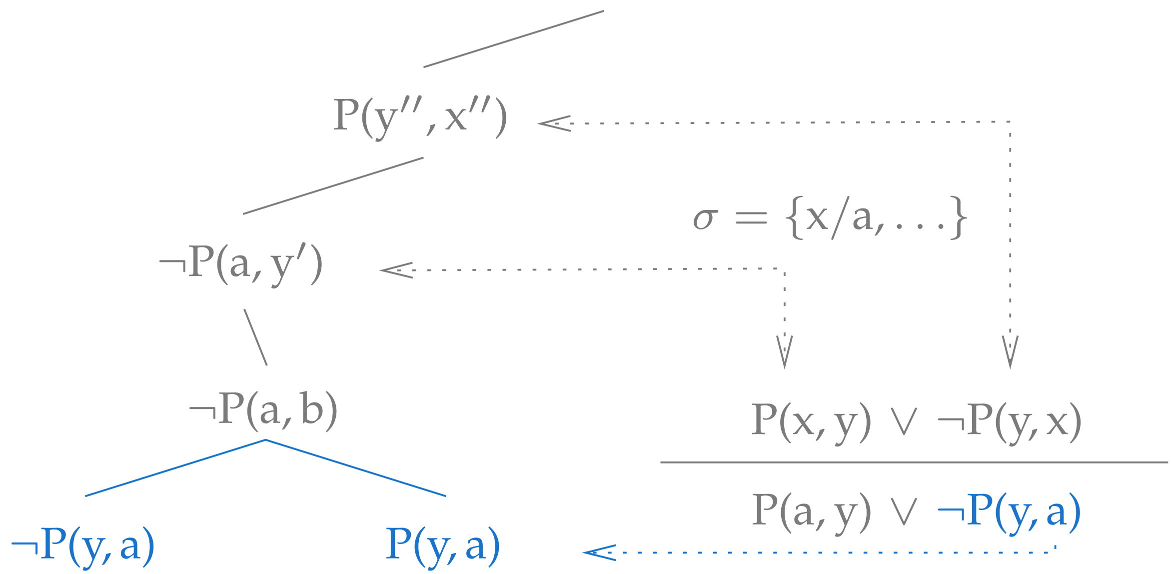
Purpose: Satisfy a clause that is currently "false"



1. Compute simultaneous most general unifier σ
2. Select from clause instance a literal not on branch
3.

Calculus: The Splitting Rule

Purpose: Satisfy a clause that is currently "false"



1. Compute simultaneous most general unifier σ
2. Select from clause instance a literal not on branch
3. Split with this literal

Calculus: The Splitting Rule

Purpose: Satisfy a clause that is currently “false”

$P(y'', x'')$

$\neg P(a, y')$

$\neg P(a, b)$ $\qquad\qquad$ $P(x, y) \vee \neg P(y, x)$

$$\{\neg P(a, c), P(c, a), \ldots\} \quad \not\models \quad P(a, c) \vee \neg P(c, a)$$

1. Compute simultaneous most general unifier σ
2. Select from clause instance a literal not on branch
3. Split with this literal

This split was really necessary!

Calculus: The Splitting Rule

Purpose: Satisfy a clause that is currently "false"

$$P(y'', x'')$$

$$\neg P(a, y')$$

$$\neg P(a, b) \qquad\qquad P(x, y) \lor \neg P(y, x)$$

$$\{\neg P(a, c), P(c, a), \ldots\} \not\models P(a, c) \lor \neg P(c, a)$$

1. Compute simultaneous most general unifier σ
2. Select from clause instance a literal not on branch
3. Split with this literal

This split was really necessary!

Proposition: If $[\![\mathcal{B}]\!] \not\models \mathcal{S}$, then split is applicable to some clause from $\mathcal{S}$

Calculus: The Splitting Rule – Another Example

Purpose: Satisfy a clause that is currently "false"

$\mathrm{P}(y'', x'')$

$\neg \mathrm{P}(a, y')$

$\neg \mathrm{P}(a, b)$

Some clause from $\mathcal{S}$

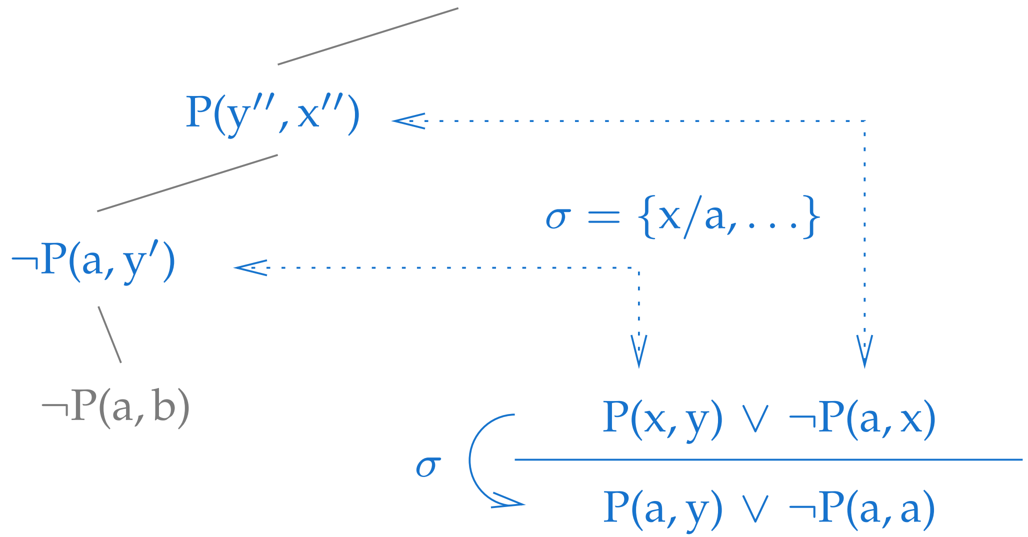
$\mathrm{P}(x, y) \vee \neg \mathrm{P}(a, x)$

1.

2.

Calculus: The Splitting Rule – Another Example

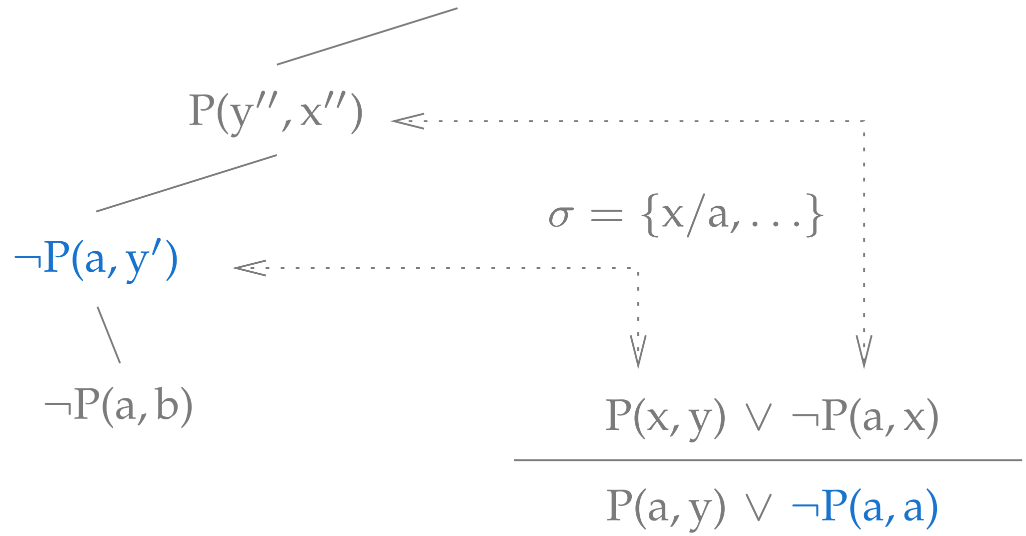
Purpose: Satisfy a clause that is currently "false"



1. Compute MGU σ of clause against branch literals
2.

Calculus: The Splitting Rule – Another Example

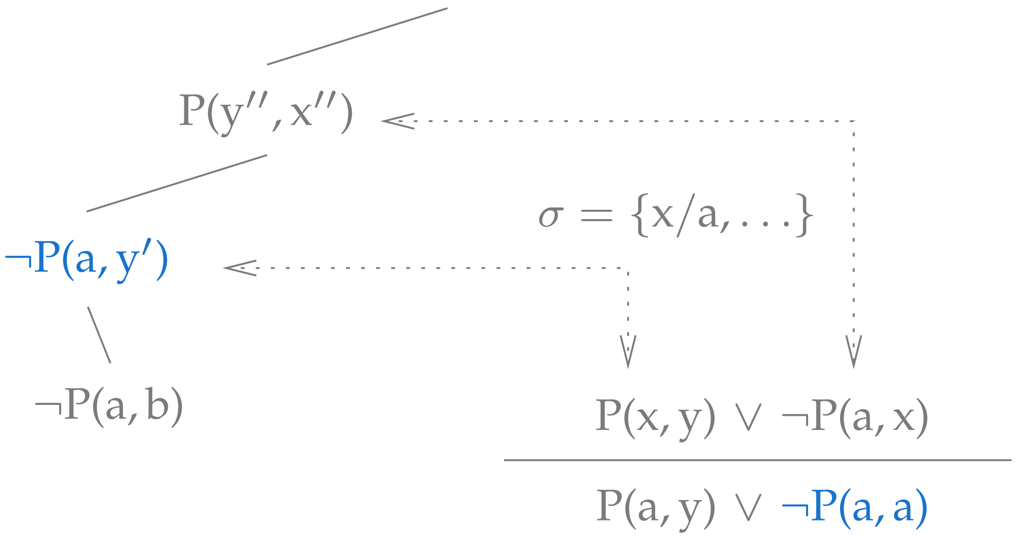
Purpose: Satisfy a clause that is currently “false”



1. Compute MGU σ of clause against branch literals
2. If clause contains “true” literal, then split is not applicable

Calculus: The Splitting Rule – Another Example

Purpose: Satisfy a clause that is currently “false”

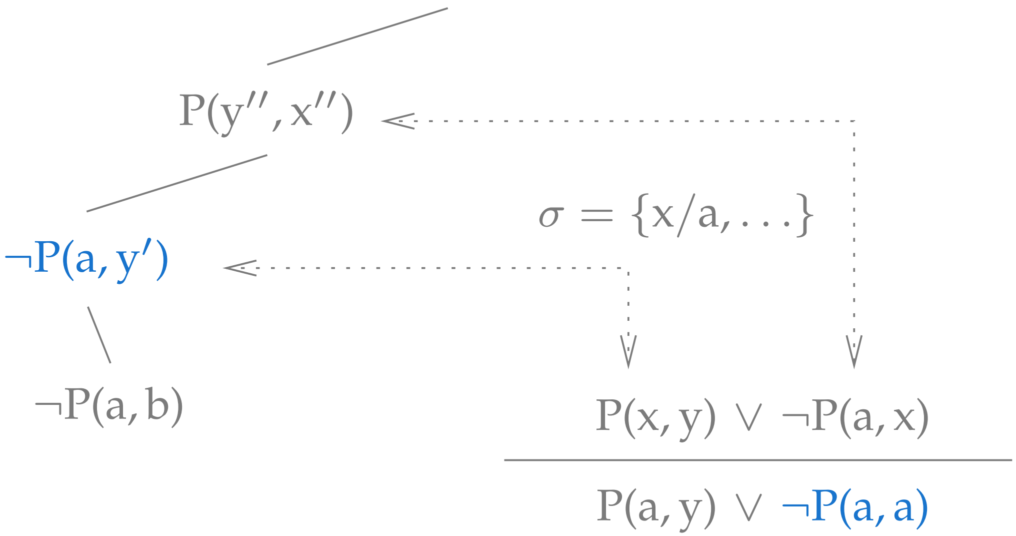


1. Compute MGU σ of clause against branch literals
2. If clause contains “true” literal, then split is not applicable

Non-applicability is a redundancy test

Calculus: The Splitting Rule – Another Example

Purpose: Satisfy a clause that is currently "false"



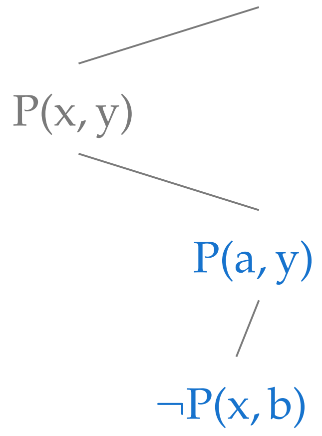
1. Compute MGU σ of clause against branch literals
2. If clause contains "true" literal, then split is not applicable

Non-applicability is a redundancy test

Proposition: If for no clause split is applicable, $[\![\mathcal{B}]\!] \models \mathcal{S}$ holds

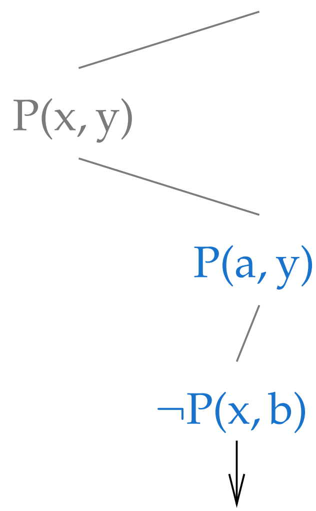
Calculus: The Commit Rule

Purpose: Achieve consistency of interpretation associated to branch



Calculus: The Commit Rule

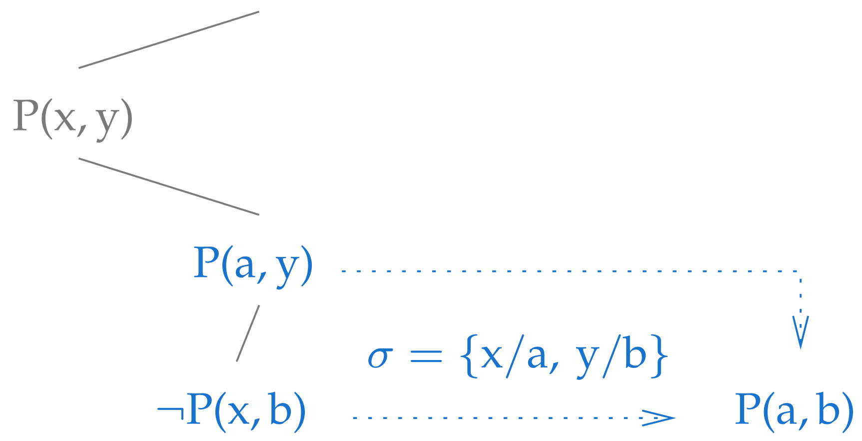
Purpose: Achieve consistency of interpretation associated to branch



Problem: $\{P(a,b), \neg P(a,b)\}$ is inconsistent!

Calculus: The Commit Rule

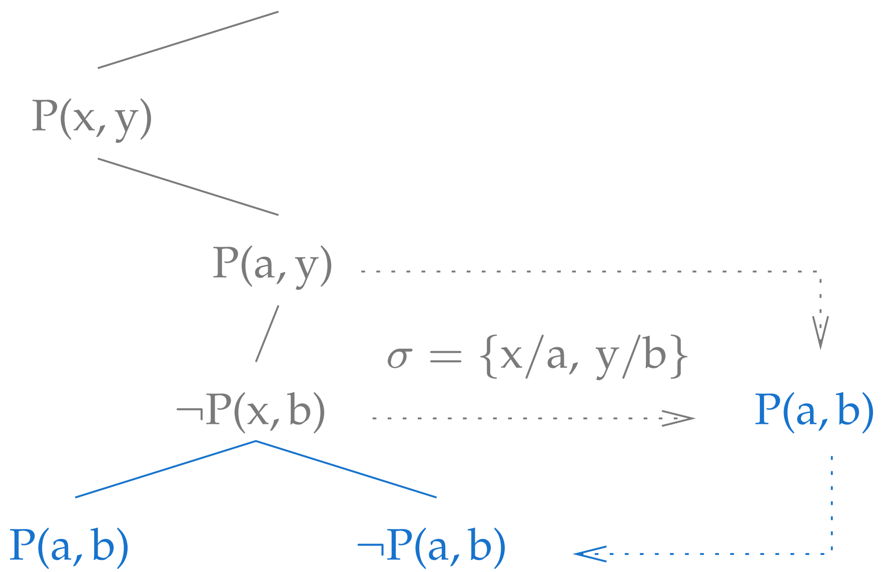
Purpose: Achieve consistency of interpretation associated to branch



1. Compute a MGU σ of branch literals with opposite sign
2.

Calculus: The Commit Rule

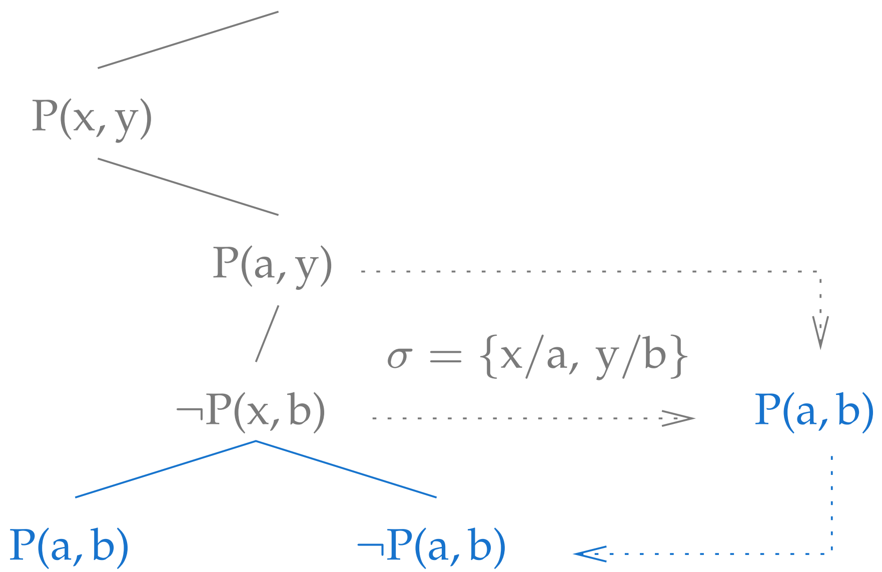
Purpose: Achieve consistency of interpretation associated to branch



1. Compute a MGU σ of branch literals with opposite sign
2. Split with instance, if not on branch

Calculus: The Commit Rule

Purpose: Achieve consistency of interpretation associated to branch



1. Compute a MGU σ of branch literals with opposite sign
2. Split with instance, if not on branch

Now have removed the inconsistency

FDPLL Complete Example

```
(1)  train(X,Y) ; flight(X,Y).          %% train from X to Y or flight from X to Y.
(2)  -flight(koblenz,X).                %% no flight from koblenz to anywhere.
(3)  flight(X,Y) :- flight(Y,X).        %% flight is symmetric.
(4)  connect(X,Y) :- flight(X,Y).       %% a flight is a connection.
(5)  connect(X,Y) :- train(X,Y).        %% a train is a connection.
(6)  connect(X,Z) :- connect(X,Y),      %% connection is a transitive relation.
                     connect(Y,Z).
```

FDPLL Complete Example

```
(1)  train(X,Y) ; flight(X,Y).         %% train from X to Y or flight from X to Y.
(2)  -flight(koblenz,X).               %% no flight from koblenz to anywhere.
(3)  flight(X,Y) :- flight(Y,X).       %% flight is symmetric.
(4)  connect(X,Y) :- flight(X,Y).      %% a flight is a connection.
(5)  connect(X,Y) :- train(X,Y).       %% a train is a connection.
(6)  connect(X,Z) :- connect(X,Y),     %% connection is a transitive relation.
                     connect(Y,Z).
```

Computed Model (as output by implementation)

```
+ flight(X, Y)
- flight(koblenz, X)
- flight(X, koblenz)
+ train(koblenz, Y)
+ train(Y, koblenz)
+ connect(X, Y)
```

FDPLL Model Computation Example - Derivation

$\langle$empyty tree$\rangle$

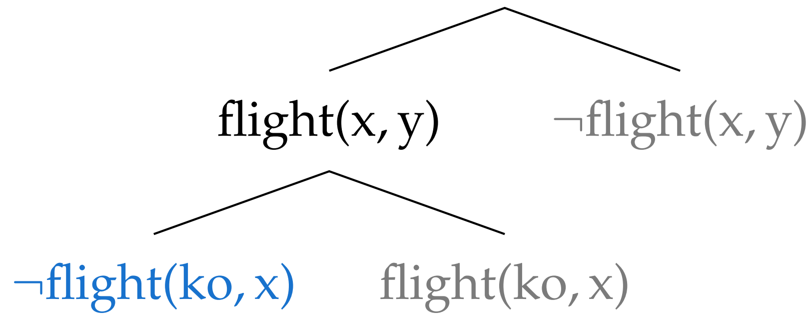
Clause instance used in inference: $\mathrm{train}(x, y) \vee \mathrm{flight}(x, y)$

FDPLL Model Computation Example - Derivation

$\mathrm{flight}(x, y) \qquad \neg\mathrm{flight}(x, y)$

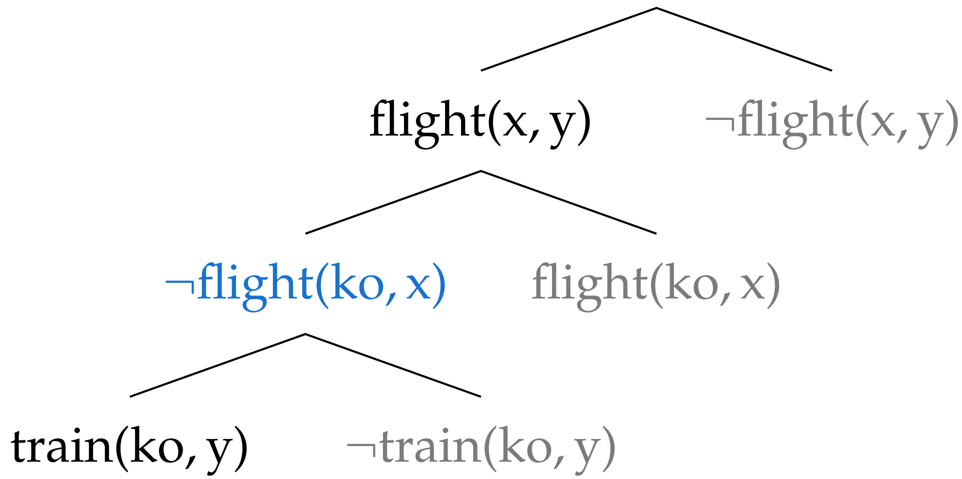
Clause instance used in inference: $\neg\mathrm{flight}(ko, x)$

FDPLL Model Computation Example - Derivation



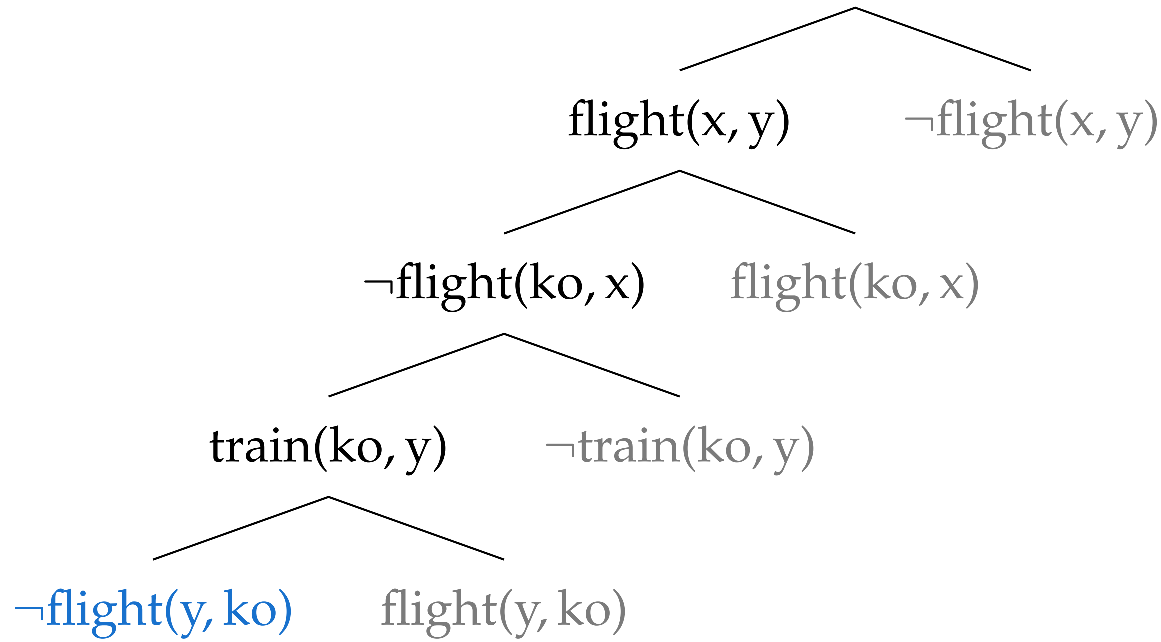
Clause instance used in inference: $\mathrm{train}(\mathrm{ko}, y) \vee \mathrm{flight}(\mathrm{ko}, y)$

FDPLL Model Computation Example - Derivation



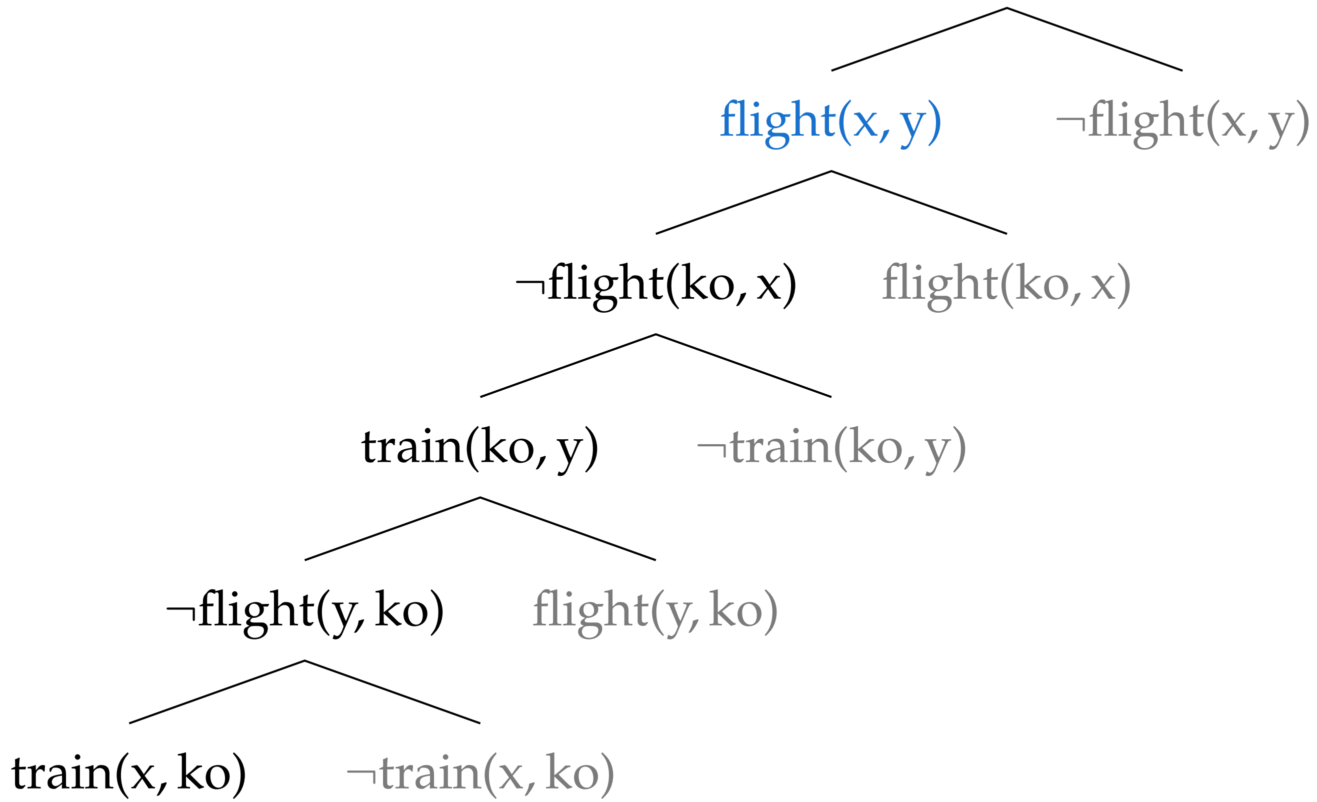
Clause instance used in inference: $\mathrm{flight}(ko, y) \lor \neg\mathrm{flight}(y, ko)$

FDPLL Model Computation Example - Derivation



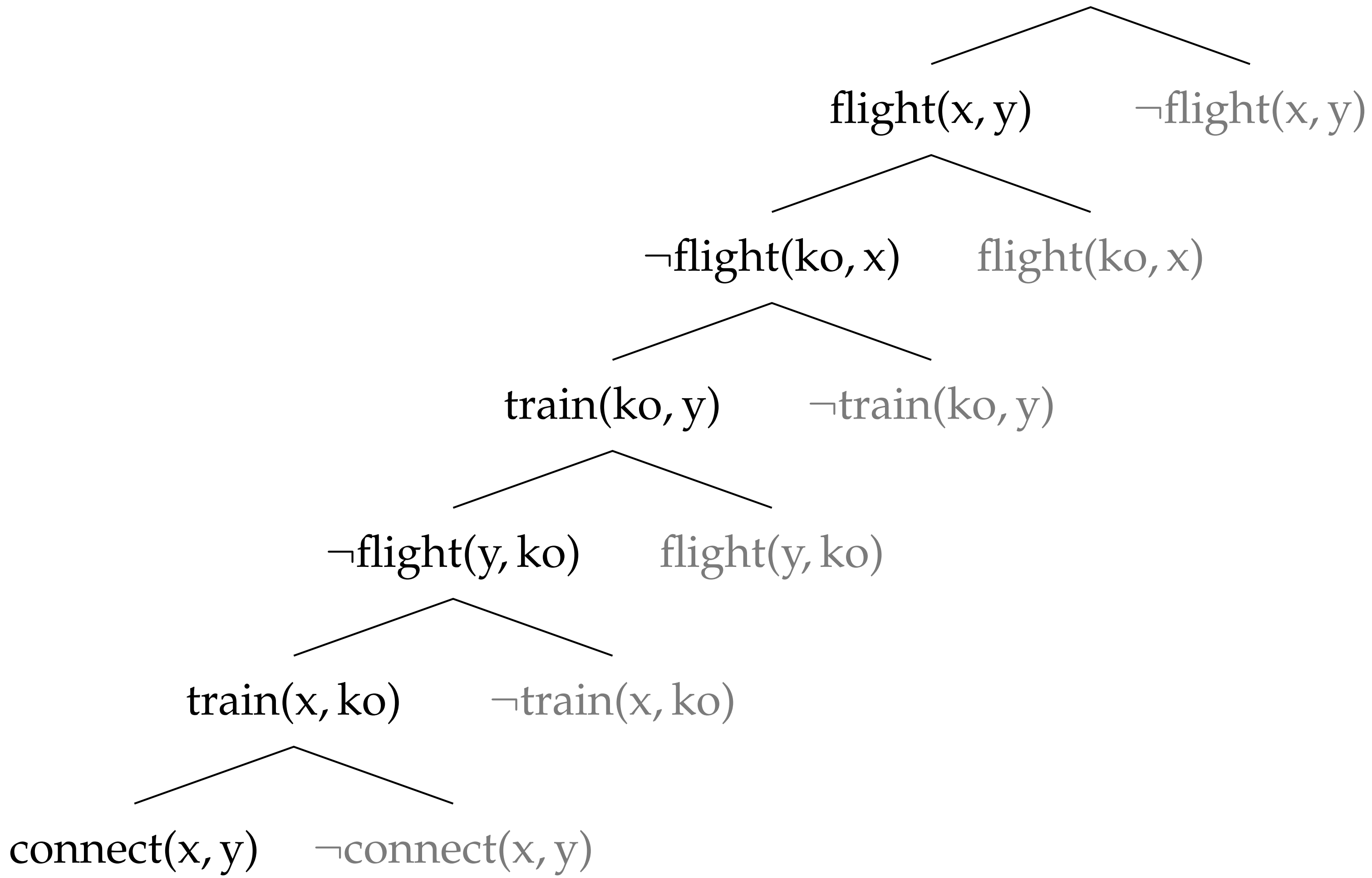
Clause instance used in inference: $\mathrm{train}(x, ko) \lor \mathrm{flight}(x, ko)$

FDPLL Model Computation Example - Derivation



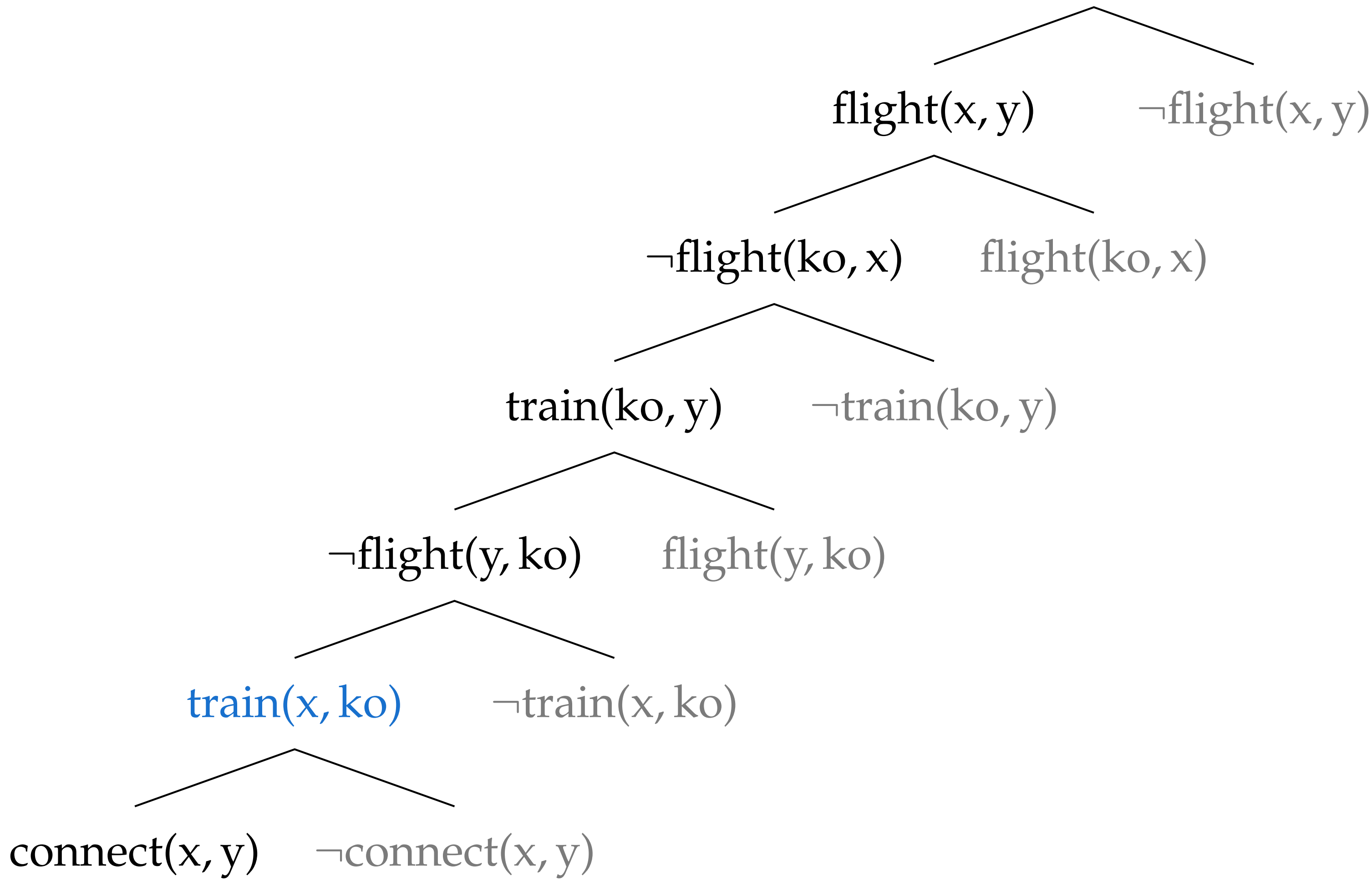
Clause instance used in inference: $\mathrm{connect}(x, y) \vee \neg\mathrm{flight}(x, y).$

FDPLL Model Computation Example - Derivation



Done. Return "satisfiable with model $\{\mathrm{flight}(x, y), \ldots, \mathrm{connect}(x, y)\}$"

FDPLL Model Computation Example - Derivation



Done. Return "satisfiable with model $\{\mathrm{flight}(x, y), \ldots, \mathrm{connect}(x, y)\}$"

Redundancy: Instance not used in inference: $\mathrm{connect}(x, ko) \vee \neg\mathrm{train}(x, ko)$

Optional Inference Rule – Universal Splits

(1) $\mathrm{P(x)}$ (2) $\neg\mathrm{P(x)} \vee \mathrm{Q(x)}$

Split based on tautology $\forall \mathrm{x}\ \mathrm{P(x)} \vee \neg\forall \mathrm{x}\ \mathrm{P(x)}$:

$\forall \mathrm{x}\ \mathrm{P(x)}$ $\qquad$ $\underset{\star}{\neg\mathrm{P(sk1)}}$

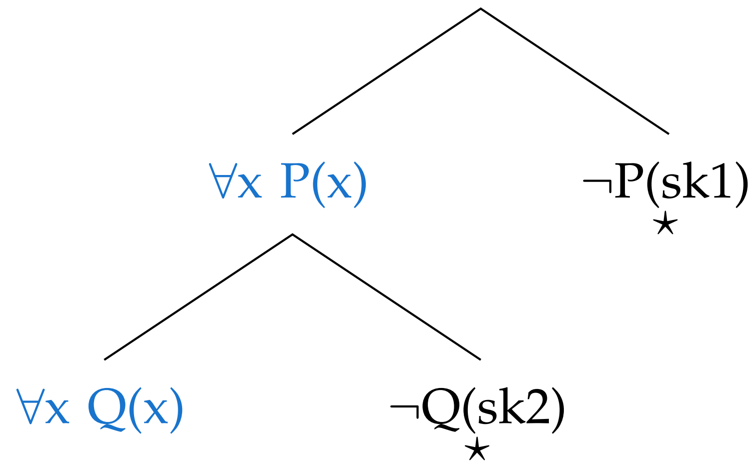
Sources for Universal Splits

- Unit input clauses

Optional Inference Rule – Universal Splits

(1) $P(x)$ (2) $\neg P(x) \vee Q(x)$

Split based on tautology $\forall x\ P(x) \vee \neg \forall x\ P(x)$:



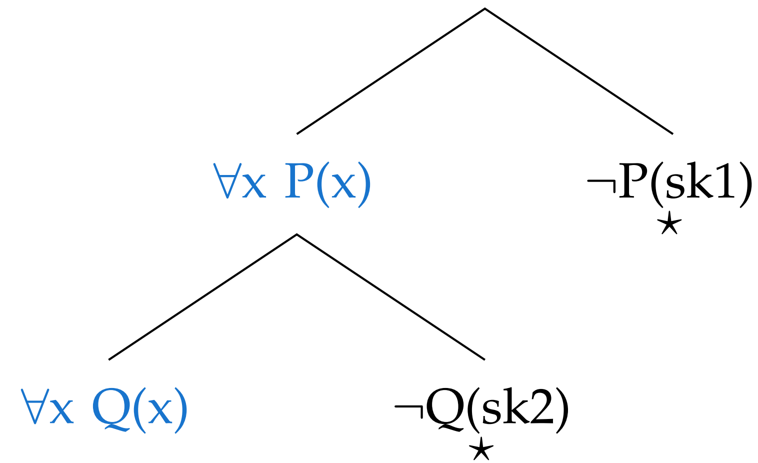
Sources for Universal Splits

- Unit input clauses
- Resolving away $n - 1$ literals from an n-literal clause (UR-Resolution)

Optional Inference Rule – Universal Splits

(1) $P(x)$ (2) $\neg P(x) \vee Q(x)$

Split based on tautology $\forall x\ P(x) \vee \neg\forall x\ P(x)$:



Sources for Universal Splits

- Unit input clauses
- Resolving away $n-1$ literals from an n-literal clause (UR-Resolution)

Advantages: – No “exceptions” permitted, hence much better efficiency
– Subsumption

Calculus: Summary / Properties

Summary

- DPLL data structure lifted to first-order logic level
- Two simple inference rules, controlled by unification
- Computes with interpretations/models
- Semantical redundancy criterion

Calculus: Summary / Properties

Summary

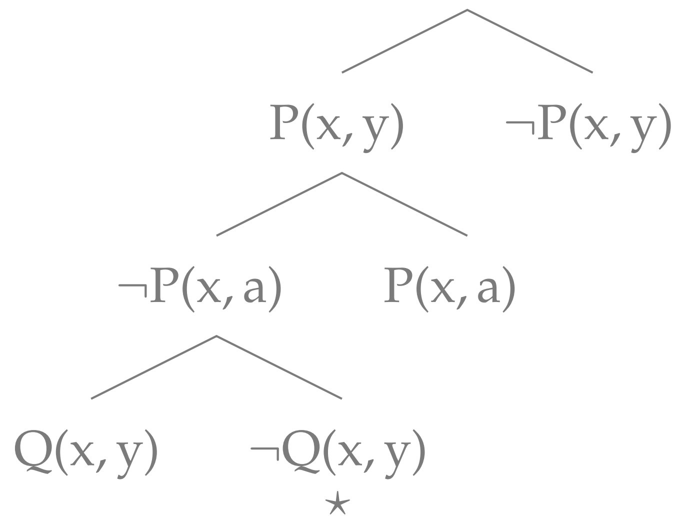
- DPLL data structure lifted to first-order logic level
- Two simple inference rules, controlled by unification
- Computes with interpretations/models
- Semantical redundancy criterion

Properties

- Soundness and completeness (with fair strategy).
- Extension: More efficient reasoning with unit clauses (e.g. $\forall x\ P(x, a)$)
- Proof convergence (avoids backtracking the semantics trees)
- Decides function-free clause logic (Bernays-Schönfinkel class)
  Covers e.g. Basic modal logic, Description logic, DataLog
  Returns model in satisfiable case
  But: Resolution better on other classes!

[Fermüller et. al. Handbook AR 2001 (e.g. Gödel class, Monadic class, Guarded Fragment,...)]

First-Order Semantic Trees



Issues:

- How are variables treated?
  (a) Universal, as in Resolution?, (b) Rigid, as in Tableaux? (c) Schema!
- How to extract an interpretation from a branch? ✔
- When is a branch closed? ✔
- How to construct such trees (calculus)? ✔

Overview

Propositional DPLL as a semantic tree method ✔

First-Order DPLL so far ✔

FDPLL ✔

Relation to other calculi

Families of First-Order Logic Calculi

Consider a transitivity clause $\mathrm{P}(x,z) \leftarrow \mathrm{P}(x,y) \wedge \mathrm{P}(y,z)$.

Families of First-Order Logic Calculi

Consider a transitivity clause $\mathrm{P}(x,z) \leftarrow \mathrm{P}(x,y) \wedge \mathrm{P}(y,z)$.

[Bachmair & Ganzinger, Handbook AR 2001], [Fermüller et. al., Handbook AR 2001]

Resolution:

$$\begin{aligned}
\mathrm{P}(x,z') &\leftarrow \mathrm{P}(x,y) \wedge \mathrm{P}(y,z) \wedge \mathrm{P}(z,z') \\
\mathrm{P}(x,z'') &\leftarrow \mathrm{P}(x,y) \wedge \mathrm{P}(y,z) \wedge \mathrm{P}(z,z') \wedge \mathrm{P}(z',z'')
\end{aligned}$$

Families of First-Order Logic Calculi

Consider a transitivity clause $\mathrm{P}(x,z) \leftarrow \mathrm{P}(x,y) \wedge \mathrm{P}(y,z)$.

[Bachmair & Ganzinger, Handbook AR 2001], [Fermüller et. al., Handbook AR 2001]

Resolution:

$$\begin{aligned}
\mathrm{P}(x,z') &\leftarrow \mathrm{P}(x,y) \wedge \mathrm{P}(y,z) \wedge \mathrm{P}(z,z') \\
\mathrm{P}(x,z'') &\leftarrow \mathrm{P}(x,y) \wedge \mathrm{P}(y,z) \wedge \mathrm{P}(z,z') \wedge \mathrm{P}(z',z'')
\end{aligned}$$

Does not terminate for function-free clause sets

Complicated to extract model

Very good on other classes, Equality

Families of First-Order Logic Calculi

Consider a transitivity clause $P(x,z) \leftarrow P(x,y) \wedge P(y,z)$.

[Bachmair & Ganzinger, Handbook AR 2001], [Fermüller et. al., Handbook AR 2001]

Resolution:

$$P(x,z') \leftarrow P(x,y) \wedge P(y,z) \wedge P(z,z')$$
$$P(x,z'') \leftarrow P(x,y) \wedge P(y,z) \wedge P(z,z') \wedge P(z',z'')$$

Does not terminate for function-free clause sets

Complicated to extract model

Very good on other classes, Equality

Rigid Variable Approaches:

FO-DPLL: [Chang&Lee 73]
Tableaux and CM: [Peltier, IGPL 99], [Baumgartner et al, CADE 99], [Beckert, FTP 2000], [Giese, CADE 01]

$$P(x',z') \leftarrow P(x',y') \wedge P(y',z')$$
$$P(x'',z'') \leftarrow P(x'',y'') \wedge P(y'',z'')$$

Families of First-Order Logic Calculi

Consider a transitivity clause $P(x,z) \leftarrow P(x,y) \wedge P(y,z)$.

[Bachmair & Ganzinger, Handbook AR 2001], [Fermüller et. al., Handbook AR 2001]

Resolution:

$$P(x,z') \leftarrow P(x,y) \wedge P(y,z) \wedge P(z,z')$$
$$P(x,z'') \leftarrow P(x,y) \wedge P(y,z) \wedge P(z,z') \wedge P(z',z'')$$

- Does not terminate for function-free clause sets
- Complicated to extract model
- Very good on other classes, Equality

FO-DPLL: [Chang&Lee 73]
Tableaux and CM: [Peltier, IGPL 99], [Baumgartner et al, CADE 99], [Beckert, FTP 2000], [Giese, CADE 01]

Rigid Variable Approaches:

$$P(x',z') \leftarrow P(x',y') \wedge P(y',z')$$
$$P(x'',z'') \leftarrow P(x'',y'') \wedge P(y'',z'')$$

- Unpredictable number of variants, weak redundancy test
- Difficult to avoid unnecessary (!) backtracking
- Difficult to extract model

Families of First-Order Logic Calculi

Consider a transitivity clause $P(x, z) \leftarrow P(x, y) \wedge P(y, z)$.

Families of First-Order Logic Calculi

Consider a transitivity clause $\mathrm{P}(x, z) \leftarrow \mathrm{P}(x, y) \wedge \mathrm{P}(y, z)$

HL [Lee&Plaisted, JAR 92], SHL [Chu&Plaisted, CADE 94], DM [Billon, TABLEAUX 96], OSHL [Plaisted & Zhu, AAAI 97], Hyper Tableaux NG [TABLEAUX 98], [van Eijck, CSL 01]

Instance Based Methods:

$$\mathrm{P}(x, z) \leftarrow \mathrm{P}(x, y) \wedge \mathrm{P}(y, z)$$
$$\mathrm{P}(a, z) \leftarrow \mathrm{P}(a, y) \wedge \mathrm{P}(y, b)$$

Families of First-Order Logic Calculi

Consider a transitivity clause $P(x,z) \leftarrow P(x,y) \wedge P(y,z)$

HL [Lee&Plaisted, JAR 92], SHL [Chu&Plaisted, CADE 94], DM [Billon, TABLEAUX 96], OSHL [Plaisted & Zhu, AAAI 97], Hyper Tableaux NG [TABLEAUX 98], [van Eijck, CSL 01]

Instance Based Methods:

$$P(x,z) \leftarrow P(x,y) \wedge P(y,z)$$
$$P(a,z) \leftarrow P(a,y) \wedge P(y,b)$$

Weak redundancy criterion (no subsumption)

Need to keep clause instances (memory problem)

Families of First-Order Logic Calculi

Consider a transitivity clause $P(x,z) \leftarrow P(x,y) \wedge P(y,z)$

HL [Lee&Plaisted, JAR 92], SHL [Chu&Plaisted, CADE 94], DM [Billon, TABLEAUX 96], OSHL [Plaisted & Zhu, AAAI 97], Hyper Tableaux NG [TABLEAUX 98], [van Eijck, CSL 01]

Instance Based Methods:

$$P(x,z) \leftarrow P(x,y) \wedge P(y,z)$$
$$P(a,z) \leftarrow P(a,y) \wedge P(y,b)$$

Weak redundancy criterion (no subsumption)

Need to keep clause instances (memory problem)

Clauses do not become longer (cf. Resolution)

May delete variant clauses (cf. Rigid Variable Approach)

Families of First-Order Logic Calculi

Consider a transitivity clause $P(x,z) \leftarrow P(x,y) \wedge P(y,z)$

HL [Lee&Plaisted, JAR 92], SHL [Chu&Plaisted, CADE 94], DM [Billon, TABLEAUX 96], OSHL [Plaisted & Zhu, AAAI 97], Hyper Tableaux NG [TABLEAUX 98], [van Eijck, CSL 01]

Instance Based Methods:

$$P(x,z) \leftarrow P(x,y) \wedge P(y,z)$$
$$P(a,z) \leftarrow P(a,y) \wedge P(y,b)$$

Weak redundancy criterion (no subsumption)

Need to keep clause instances (memory problem)

Clauses do not become longer (cf. Resolution)

May delete variant clauses (cf. Rigid Variable Approach)

FDPLL in this tradition, but

- need not keep instances

Families of First-Order Logic Calculi

Consider a transitivity clause $P(x,z) \leftarrow P(x,y) \wedge P(y,z)$

HL [Lee&Plaisted, JAR 92], SHL [Chu&Plaisted, CADE 94], DM [Billon, TABLEAUX 96], OSHL [Plaisted & Zhu, AAAI 97], Hyper Tableaux NG [TABLEAUX 98], [van Eijck, CSL 01]

Instance Based Methods:

$$
\begin{aligned}
P(x,z) &\leftarrow P(x,y) \wedge P(y,z) \\
P(a,z) &\leftarrow P(a,y) \wedge P(y,b)
\end{aligned}
$$

Weak redundancy criterion (no subsumption)

Need to keep clause instances (memory problem)

Clauses do not become longer (cf. Resolution)

May delete variant clauses (cf. Rigid Variable Approach)

FDPLL in this tradition, but

- need not keep instances
- conceptually different: – binary splitting
  – based on first-order interpretations

Overview

Propositional DPLL as a semantic tree method ✔

First-Order DPLL so far ✔

FDPLL ✔

Relation to other calculi ✔

Conclusions

Implementation

- In Eclipse Prolog, $\approx$ 1300 LoC, `http://www.uni-koblenz.de/~peter/FDPLL/`

Conclusions

Implementation

- In Eclipse Prolog, $\approx$ 1300 LoC, `http://www.uni-koblenz.de/~peter/FDPLL/`
- Some improvements (Dependency directed backtracking)
  Still slow due to low inference rate and non-optimal algorithm

Conclusions

Implementation

- In Eclipse Prolog, $\approx$ 1300 LoC, `http://www.uni-koblenz.de/~peter/FDPLL/`
- Some improvements (Dependency directed backtracking)
  Still slow due to low inference rate and non-optimal algorithm

. . . on TPTP

Conclusions

Implementation

- In Eclipse Prolog, $\approx$ 1300 LoC, `http://www.uni-koblenz.de/~peter/FDPLL/`
- Some improvements (Dependency directed backtracking)
  Still slow due to low inference rate and non-optimal algorithm

. . . on TPTP

- Solves some moderately difficult problems e.g. IVT
  3-move Rubik's cube problem (6^{54} possible state predicate instances)

Conclusions

Implementation

- In Eclipse Prolog, $\approx$ 1300 LoC, `http://www.uni-koblenz.de/~peter/FDPLL/`
- Some improvements (Dependency directed backtracking)
  Still slow due to low inference rate and non-optimal algorithm

... on TPTP

- Solves some moderately difficult problems e.g. IVT
  3-move Rubik's cube problem (6^{54} possible state predicate instances)
- Relative strength: non-Horn, satisfiable problems (Modal Logic)

Conclusions

Implementation

- In Eclipse Prolog, $\approx$ 1300 LoC, `http://www.uni-koblenz.de/~peter/FDPLL/`
- Some improvements (Dependency directed backtracking)
  Still slow due to low inference rate and non-optimal algorithm

. . . on TPTP

- Solves some moderately difficult problems e.g. IVT
  3-move Rubik's cube problem (6^{54} possible state predicate instances)
- Relative strength: non-Horn, satisfiable problems (Modal Logic)
- Success rates:
  State-of-the-art systems: $\approx$ 55%, FDPLL: $\approx$ 40%
  State-of-the-art Resolution systems, 70's technology: $\approx$ 30%

Conclusions

Summary

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques
- Directly lifts propositional DPLL to first-order level

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques
- Directly lifts propositional DPLL to first-order level
- New concept: Schema variables

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques
- Directly lifts propositional DPLL to first-order level
- New concept: Schema variables
- Sound and complete

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques
- Directly lifts propositional DPLL to first-order level
- New concept: Schema variables
- Sound and complete

Outlook

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques
- Directly lifts propositional DPLL to first-order level
- New concept: Schema variables
- Sound and complete

Outlook

- Nonmonotonic logic variant (document management application)

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques
- Directly lifts propositional DPLL to first-order level
- New concept: Schema variables
- Sound and complete

Outlook

- Nonmonotonic logic variant (document management application)
- Improve model building capabilities

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques
- Directly lifts propositional DPLL to first-order level
- New concept: Schema variables
- Sound and complete

Outlook

- Nonmonotonic logic variant (document management application)
- Improve model building capabilities
- Theory reasoning (Equality, Decision procedures)

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques
- Directly lifts propositional DPLL to first-order level
- New concept: Schema variables
- Sound and complete

Outlook

- Nonmonotonic logic variant (document management application)
- Improve model building capabilities
- Theory reasoning (Equality, Decision procedures)
- Combine with other techniques (OSHL)

Conclusions

Summary

- Motivation: combine successful propositional and first-order techniques
- Directly lifts propositional DPLL to first-order level
- New concept: Schema variables
- Sound and complete

Outlook

- Nonmonotonic logic variant (document management application)
- Improve model building capabilities
- Theory reasoning (Equality, Decision procedures)
- Combine with other techniques (OSHL)
- Serious implementation (OCaml)