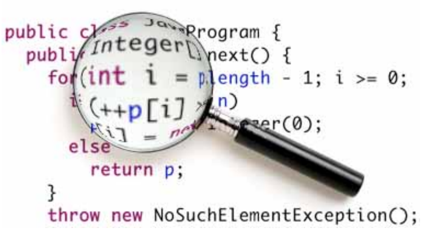
15-819M: Data, Code, Decisions

14: Instance Based Methods

André Platzer

aplatzer@cs.cmu.edu Carnegie Mellon University, Pittsburgh, PA



Recent Trends in Instance Based Proving

Instance Based Methods (IMs): a family of calculi and proof procedures for first-order logic (clauses), developed over past 15 years.

Overview

- Common principles behind IMs, some calculi, proof procedures
- Comparison among IMs, difference from tableaux and resolution
- Ranges of applicability/non-applicability
- Picking up SAT techniques

? Improvements and extensions: universal variables, equality, . . .

? Implementations and implementation techniques

Acknowledgments

Slides based on tutorial“Instance Based Methods” by Peter Baumgartner and Gernot Stenz at TABLEAUX’05

The Theory Strikes Back

Skolem-Herbrand-Löwenheim Theorem

$\forall\phi$ is unsatisfiable iff some finite set of ground instances $\{\phi\gamma_1, \ldots, \phi\gamma_n\}$ is unsatisfiable

For refutational theorem proving (i.e. start with negated conjecture) thus sufficient to

- incrementally enumerate finite sets of ground instances, and
- test each for propositional unsatisfiability.
  Stop with "unsatisfiable" when the first propositionally unsatisfiability set arrives

This has been known for a long time: Gilmore's algorithm, DPLL
It is also a common principle behind IMs

The Theory Strikes Back

Skolem-Herbrand-Löwenheim Theorem

$\forall\phi$ is unsatisfiable iff some finite set of ground instances $\{\phi\gamma_1, \ldots, \phi\gamma_n\}$ is unsatisfiable

For refutational theorem proving (i.e. start with negated conjecture) thus sufficient to

- incrementally enumerate finite sets of ground instances, and
- test each for propositional unsatisfiability.
  Stop with "unsatisfiable" when the first propositionally unsatisfiability set arrives

This has been known for a long time: Gilmore's algorithm, DPLL
It is also a common principle behind IMs

So what's special about IMs? Do this in a clever way!

An early IM: the DPLL Procedure

Preprocessing:

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

Outer loop:
Grounding

Inner loop:
Propositional
DPLL

An early IM: the DPLL Procedure

Preprocessing:

Given Formula

$\forall x\, \exists y\, P(y, x)$
$\land \forall z\, \neg P(z, a)$

→

Clause Form

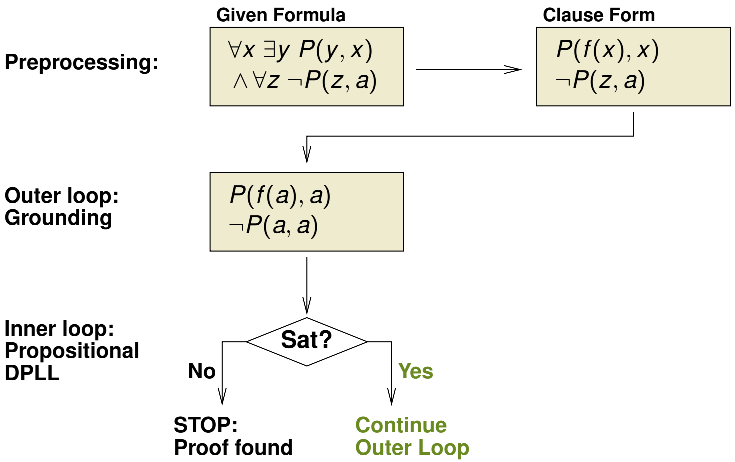
$P(f(x), x)$
$\neg P(z, a)$

Outer loop: Grounding

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

Inner loop: Propositional DPLL

An early IM: the DPLL Procedure



An early IM: the DPLL Procedure

Preprocessing:

Given Formula

$\forall x\, \exists y\, P(y, x)$
$\land \forall z\, \neg P(z, a)$

Clause Form

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

Outer loop: Grounding

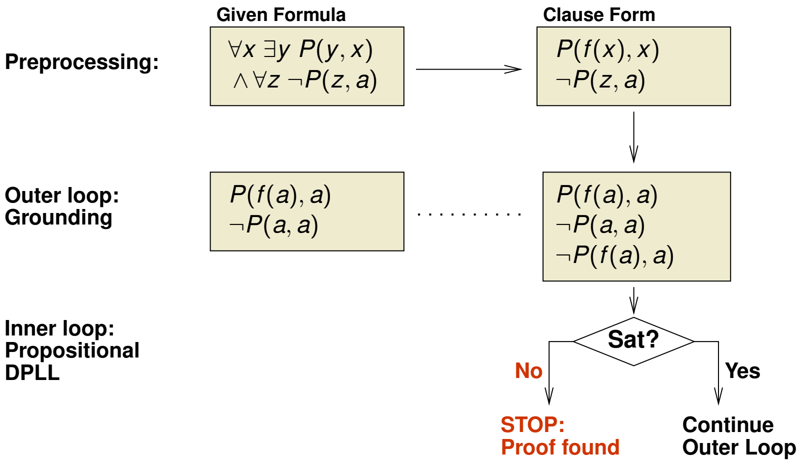
$P(f(a), a)$
$\neg P(a, a)$

. . . .

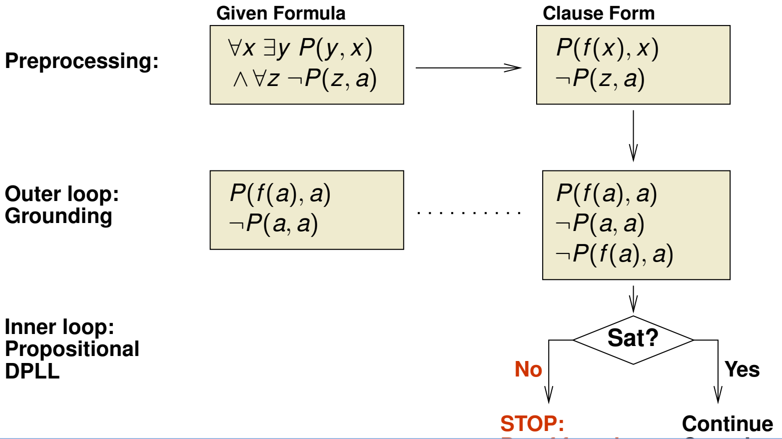
$P(f(a), a)$
$\neg P(a, a)$
$\neg P(f(a), a)$

Inner loop: Propositional DPLL

An early IM: the DPLL Procedure



An early IM: the DPLL Procedure



Problems/Issues

- Controlled grounding process in *outer loop* (irrelevant instances)
- Repeat work *across* inner loops
- Weak redundancy criterion *within* inner loop

Part I: Overview of IMs

- Classification of IMs and some representative calculi
- Emphasis not too much on the details
- Identify common principles and also differences
- Comparison with resolution and tableaux
- Applicability/Non-Applicability

Development of IMs (I)

IM History

- List existing methods (apologies for "forgotten" ones ...)
- Define abbreviations used later on
- Provide pointer to literature
- Itemize structure indicates reference relation (when obvious)
- *Not:* table of contents of what follows
  (presentation is systematic instead of historical)

DPLL – Davis-Putnam-Logemann-Loveland procedure [Davis and Putnam, 1960], [Davis *et al.*, 1962b], [Davis *et al.*, 1962a], [Davis, 1963], [Chinlund *et al.*, 1964]

FDPLL – First-Order DPLL [Baumgartner, 2000]

- ME – Model Evolution Calculus [Baumgartner and Tinelli, 2003]
- ME with Equality [Baumgartner and Tinelli, 2005]

Development of IMs (III)

HL – Hyperlinking [Lee and Plaisted, 1992]

- SHL – Semantic Hyper Linking [Chu and Plaisted, 1994]
- OSHL – Ordered Semantic Hyper Linking [Plaisted and Zhu, 1997]

PPI – Primal Partial Instantiation (1994) [Hooker *et al.*, 2002]

- "Inst-Gen" [Ganzinger and Korovin, 2003]

MACE-Style Finite Model Buiding [McCune, 1994],..., [Claessen and Sörensson, 2003]

DC – Disconnection Method [Billon, 1996]

- HTNG - Hyper Tableaux Next Generation [Baumgartner, 1998]
- DCTP – Disconnection Tableaux [Letz and Stenz, 2001]

Ginsberg & Parkes method [Ginsberg and Parkes, 2000]

OSHT – Ordered Semantic Hyper Tableaux [Yahya and Plaisted, 2002]

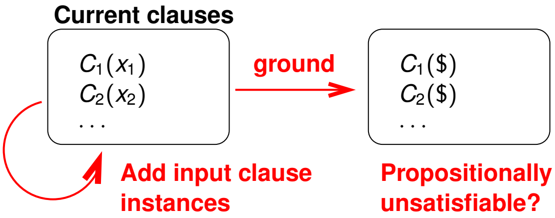
Two-Level vs. One-Level Calculi

Two-Level Calculi

- Separation between instance generation and SAT solving phase
- Uses (arbitrary) propositional SAT solver as a subroutine
- DPLL, HL, SHL, OSHL, PPI, Inst-Gen

Problem:

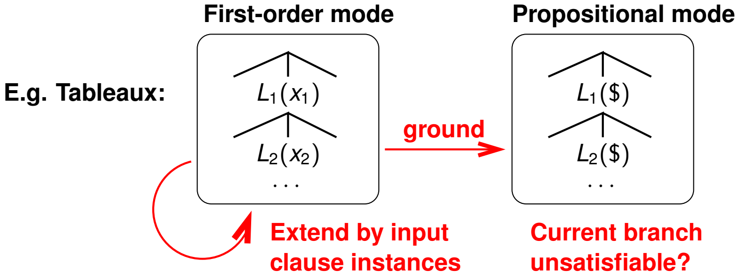
How to tell SAT solver e.g. $\forall x P(x)$?



Two-Level vs. One-Level Calculi

One-Level Calculi

- Monolithic: one single base calculus, two modes of operation
- First-order mode: base calculus clauses from input instances
- Propositional mode: $-instance of clauses drives first-order mode
- HyperTableaux NG, DCTP (see Part II), OSHT, FDPLL, ME

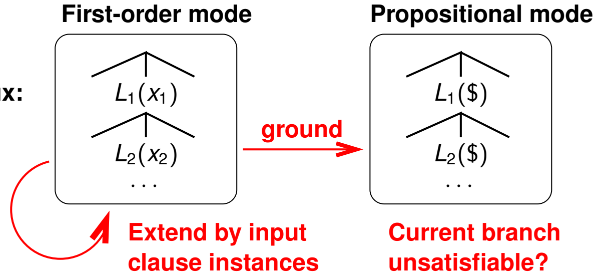


Two-Level vs. One-Level Calculi

One-Level Calculi

- Monolithic: one single base calculus, two modes of operation
- First-order mode: base calculus clauses from input instances
- Propositional mode: $-instance of clauses drives first-order mode
- HyperTableaux NG, DCTP (see Part II), OSHT, FDPLL, ME

E.g. Tableaux:



Next: two-level calculus “Inst-Gen”

Inst-Gen

- Inst-Gen is simple and elegant
- Next:
    - Idea behind Inst-Gen
      (it provides a clue to the working of two-level calculi)
    - Inst-Gen calculus
    - Comparison to resolution
    - Mentioning some improvements "idea behind"
- References: [Ganzinger and Korovin, 2003]

Inst-Gen - Underlying Idea (I)

Important notation:

$\perp$ denotes both a unique constant and a substitution that maps every *variable* to $\perp$.

Example (S is "current clause set"):

$$S: \quad P(x,y) \vee P(y,x) \qquad\qquad S\perp: \quad P(\perp,\perp) \vee P(\perp,\perp)$$
$$\neg P(x,x) \qquad\qquad \neg P(\perp,\perp)$$

Analyze $S\perp$:
Case 1: SAT detects unsatisfiability of $S\perp$
Then Conclude S is unsatisfiable

Inst-Gen - Underlying Idea (I)

Important notation:

$\bot$ denotes both a unique constant and a substitution that maps every *variable* to $\bot$.

Example (S is "current clause set"):

$$S: \quad P(x,y) \lor P(y,x) \qquad\qquad S\bot: \quad P(\bot,\bot) \lor P(\bot,\bot)$$
$$\neg P(x,x) \qquad\qquad \neg P(\bot,\bot)$$

Analyze $S\bot$:
Case 1: SAT detects unsatisfiability of $S\bot$
Then Conclude S is unsatisfiable

But what if $S\bot$ is satisfied by some model, denoted by $I_\bot$?

Inst-Gen - Underlying Idea (II)

Main idea:

Associate to model $I_\perp$ of $S\perp$ a *candidate model* I_S of S.
Calculus goal: add instances to S so that I_S becomes a model of S

Example:

$$S: \quad \underline{P(x)} \vee Q(x) \qquad\qquad S\perp: \quad \underline{P(\perp)} \vee Q(\perp)$$
$$\underline{\neg P(a)} \qquad\qquad\qquad\qquad \underline{\neg P(a)}$$

Analyze $S\perp$:
Case 2: SAT detects model $I_\perp = \{P(\perp), \neg P(a)\}$ of $S\perp$
Case 2.1: candidate model $I_S = \{\neg P(a)\}$ derived from literals selected in S by $I_\perp$ is not a model of S

Inst-Gen - Underlying Idea (II)

Main idea:

Associate to model $I_\perp$ of $S\perp$ a *candidate model* I_S of S.
Calculus goal: add instances to S so that I_S becomes a model of S

Example:

$$S: \quad \underline{P(x)} \lor Q(x) \qquad\qquad S\perp: \quad \underline{P(\perp)} \lor Q(\perp)$$
$$\underline{\neg P(a)} \qquad\qquad\qquad\qquad \underline{\neg P(a)}$$

Analyze $S\perp$:
Case 2: SAT detects model $I_\perp = \{P(\perp), \neg P(a)\}$ of $S\perp$
Case 2.1: candidate model $I_S = \{\neg P(a)\}$ derived from literals selected in S by $I_\perp$ is not a model of S

Add "problematic" instance $P(a) \lor Q(a)$ to S to refine I_S

Inst-Gen - Underlying Idea (III)

Clause set after adding $P(a) \vee Q(a)$

$$S: \quad \underline{P(x)} \vee Q(x) \qquad\qquad S\bot: \quad \underline{P(\bot)} \vee Q(\bot)$$
$$\quad P(a) \vee \underline{Q(a)} \qquad\qquad\qquad P(a) \vee \underline{Q(a)}$$
$$\quad \underline{\neg P(a)} \qquad\qquad\qquad\qquad \underline{\neg P(a)}$$

Analyze $S\bot$:

Case 2: SAT detects model $I_\bot = \{P(\bot), Q(a), \neg P(a)\}$ of $S\bot$

Case 2.2: candidate model $I_S = \{Q(a), \neg P(a)\}$ derived from literals <u>selected</u> in S by $I_\bot$ *is* a model of S
Then conclude S is satisfiable

Inst-Gen - Underlying Idea (III)

Clause set after adding $P(a) \vee Q(a)$

$$S: \quad \underline{P(x)} \vee Q(x) \qquad S\bot: \quad \underline{P(\bot)} \vee Q(\bot)$$
$$\quad P(a) \vee \underline{Q(a)} \qquad \quad P(a) \vee \underline{Q(a)}$$
$$\quad \underline{\neg P(a)} \qquad \quad \underline{\neg P(a)}$$

Analyze $S\bot$:

Case 2: SAT detects model $I_\bot = \{P(\bot), Q(a), \neg P(a)\}$ of $S\bot$

Case 2.2: candidate model $I_S = \{Q(a), \neg P(a)\}$ derived from literals selected in S by $I_\bot$ *is* a model of S

Then conclude S is satisfiable

How to derive candidate model I_S?

Inst-Gen - Model Construction

It provides (partial) interpretation for S_{ground} for given clause set S

$$S: \quad \begin{array}{l} \underline{P(x)} \lor Q(x) \\ P(a) \lor \underline{Q(a)} \\ \underline{\neg P(a)} \end{array} \qquad \Sigma = \{a, b\},\ S_{\text{ground}}: \quad \begin{array}{l} \underline{P(b)} \lor Q(b) \\ P(a) \lor \underline{Q(a)} \\ \underline{\neg P(a)} \end{array}$$

- For each $C_{\text{ground}} \in S_{\text{ground}}$ find *most specific* $C \in S$ that can be instantiated to C_{ground}
- Select literal in C_{ground} corresponding to selected literal in that C
- Add selected literal of that C_{ground} to I_S if not in conflict with I_S

Thus, $I_S = \{P(b), Q(a), \neg P(a)\}$

Inst-Gen - Summary so far

- Previous slides showed the main ideas underlying the working of calculus - not the calculus itself
- The models $I_\perp$ and the candidate model I_S are not needed in the calculus, but justify improvements
- And they provide the conceptual tool for the completeness proof: as instances of clauses are added, the initial approximation of a model of S is refined more and more
- The purpose of this refinement is to remove conflicts "A – $\neg A$" by selecting different literals in instances of clauses
- If this process does not lead to a refutation, every ground instance $C\gamma$ of a clause $C \in S$ will be assigned true by some sufficiently developed candidate model

Inst-Gen Inference Rule

$$\text{Inst-Gen}\ \frac{C \lor L \qquad \overline{L'} \lor D}{(C \lor L)\theta \quad (\overline{L'} \lor D)\theta} \quad \text{where}$$

(i) $\theta = \text{mgu}(L, L')$, and

(ii) θ a *proper instantiator:* maps some variables to nonvariable terms

Example:

$$\text{Inst-Gen}\ \frac{Q(x) \lor P(x,b) \quad \neg P(a,y) \lor R(y)}{Q(a) \lor P(a,b) \quad \neg P(a,b) \lor R(b)} \quad \text{where}$$

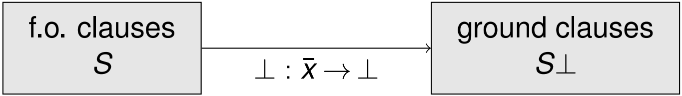
(i) $\theta = \text{mgu}(P(x,b), \neg P(a,y)) = \{x \to a, y \to b\}$, and

(ii) θ a proper instantiator

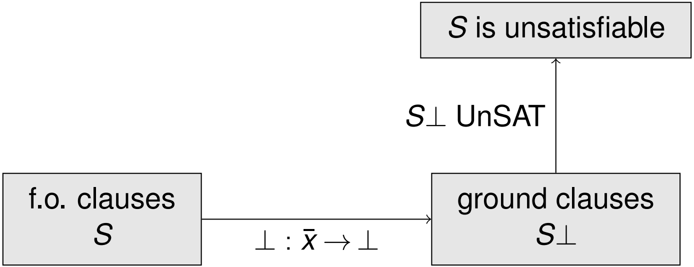
Inst-Gen - Outer Loop

f.o. clauses
S

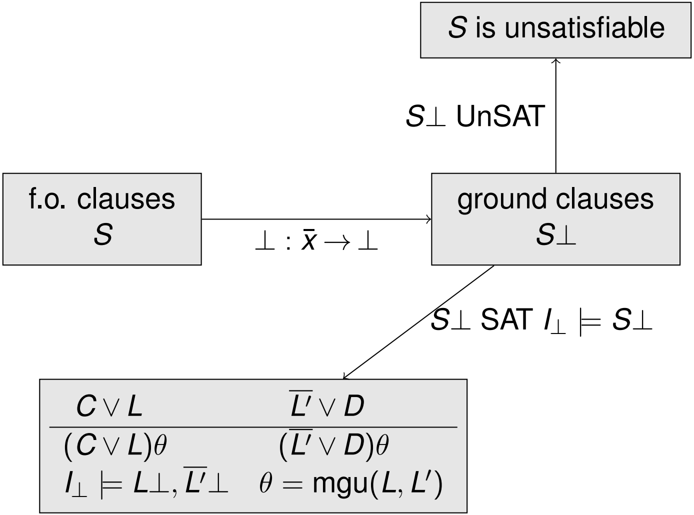
Inst-Gen - Outer Loop



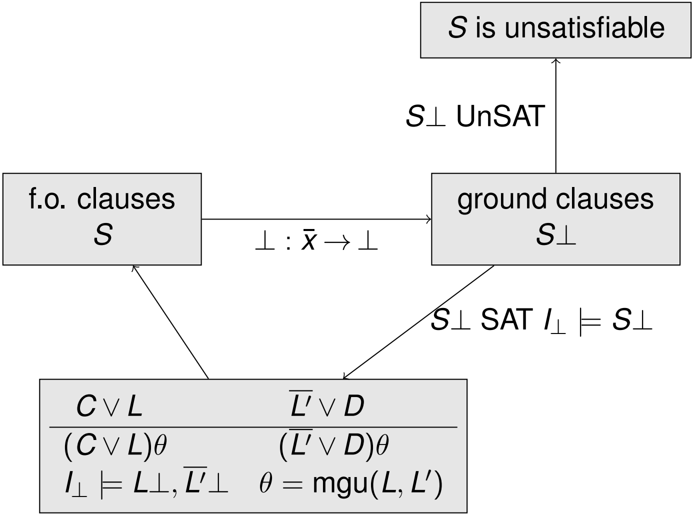
Inst-Gen - Outer Loop



Inst-Gen - Outer Loop



Inst-Gen - Outer Loop



Properties and Improvements

- As efficient as possible in propositional case
- Literal selection *in the calculus*
  - Require "back channel" from SAT solver (output of models) to select literals in S (as obtained in $I_\perp$)
  - Restrict inference rule application to selected literals
  - Need only consider instances falsified in I_S
  - Allows to extract model if S is finitely saturated
  - Flexibility: may change models $I_\perp$ arbitrarily during derivation
- Hyper-type inference rule, similar to Hyper Linking [Lee and Plaisted, 1992]
- Subsumption deletion by proper subclauses
- Special variables: allows to replace SAT solver by solver for richer fragment (guarded fragment, two-variable fragment)

Resolution vs. Inst-Gen

Resolution

$$\frac{(C \vee L) \qquad (\overline{L'} \vee D)}{(C \vee D)\theta}$$
$$\theta = \text{mgu}(L, L')$$

- Inefficient for propositional
- Length of clauses grow fast
- Recombination of clauses
- Subsumption deletion
- A-Ordered resolution: selection by term ordering
- Difficult to extract model
- Decides guarded fragment, two-variable fragment, some classes defined by Leitsch et al., not Bernays-Schönfinkel

Inst-Gen

$$\frac{C \vee L \qquad \overline{L'} \vee D}{(C \vee L)\theta \quad (\overline{L'} \vee D)\theta}$$
$$\theta = \text{mgu}(L, L')$$

- Efficient in propositional case
- Length of clauses fixed
- No recombination of clauses
- Subsumption deletion limited
- Selection based on propositional model
- Easy to extract model
- Decides Bernays-Schönfinkel class, nothing else known yet
- Current CASC-winning

Other Two-Level Calculi (I)

DPLL - Davis-Putnam-Logemann-Loveland Procedure

- Weak concept of redundancy already present (purity deletion)

PPI – Primal Partial Instantiation

- Comparable to Inst-Gen, but see [Jacobs and Waldmann, 2005]
- With fixed iterative deepening over term-depth bound

MACE-Style Finite Model Buiding (Different Focus)

- Enumerate finite domains $\{0\}$, $\{0, 1\}$, $\{0, 1, 2\}$, $\ldots$
- Transform clause set to encode search for finite domain model
- Apply (incremental) SAT solver
- Complete for finite models, not refutationally complete

Other Two-Level Calculi (II) - HL and SHL

HL - Hyper Linking (Clause Linking)

- Uses hyper type of inference rule, based on simultaneous mgu of nucleus and electrons
- Doesn't use selection (no guidance from propositional model)

SHL - Semantic Hyper Linking

- Uses "back channel" from SAT solver to guide search: find *single* ground clause $C\gamma$ so that $I_\perp \not\models C\gamma$ and add it
- Doesn't use unification; basically guess ground instance, but ...
- Practical effectiveness achieved by other devices:
  - Start with "natural" initial interpretation
  - "Rough resolution" to eliminate "large" literals
  - Predicate replacement to unfold definitions [Lee and Plaisted, 1989]
- Important reference: [Plaisted, 1994]

Other Two-Level Calculi (III) - OSHL

OSHL - Ordered Semantic Hyper Linking

- [Plaisted and Zhu, 1997], [Plaisted and Zhu, 2000]
- Goal-orientation by chosing "natural" initial interpretation I_0 that falsifies (negated) theorem clause, but satisfies most of the theory clauses
- Stepwise modify I_0
  Modified interpretation represented as $I_0(L_1, \ldots, L_m)$
  (which is like I_0 except for ground literals $L_1, \ldots, L_m$)
- Completeness via fair enumeration of modifications
- Special treatment of unit clauses
- Subsumption by proper subclauses
- Uses A-ordered resolution as propositional decision procedure

OSHL Proof Procedure

Input: S, I_0 ;; S input clauses I_0 initial interpretation
$I := I_0$;; Current interpretation
$G := \{\}$;; Current ground instances from S
while $\{\} \notin G$ **do**
  if $I \models S$;; ... and this can be detected
    then return "satisfiable"
  search $C \in S$ and γ
    such that $I \not\models C\gamma$;; Instance generation
  $G := simplify(G, C\gamma)$;; Have $C\gamma \in G$ after simplification
  $I := update(I_0, G)$;; Update such that $I \models G$
end while
return "unsatisfiable"

How to search C and γ for given $I = I_0(L_1, \ldots, L_m)$

- Guess $C \in S$ and partition $C = C_1 \cup C_2$
- Let θ matcher of C_1 to $(\overline{L_1}, \ldots, \overline{L_m})$ (with complementary signs)
- Guess δ s.th. $I_0(L_1, \ldots, L_m) \not\models C\gamma$, where $\gamma = \theta\delta$

Search and Update in OSHL

$I_o = \{R(a)\}$
(all other atoms false)

S:

(1)	$R(a) \leftarrow$	(4)	$\leftarrow Q(a,c)$
(2)	$P(x) \leftarrow R(a)$	(5)	$\leftarrow R(c)$
(3)	$R(y) \vee Q(x,y) \leftarrow P(x)$		

OSHL Refutation:

(2)	I_0	$\not\models$	$P(x) \leftarrow R(a)$
	I_0	$\not\models$	$P(a) \leftarrow R(a)$
(3)	$I_0(P(a))$	$\not\models$	$R(y) \vee Q(x,y) \leftarrow P(x)$
	$I_0(P(a))$	$\not\models$	$R(y) \vee Q(a,y) \leftarrow P(a)$
	$I_0(P(a))$	$\not\models$	$R(c) \vee Q(a,c) \leftarrow P(a)$
(5)	$I_0(P(a), R(c))$	$\not\models$	$\leftarrow R(c)$
(4)	$I_0(P(a), Q(a,c))$	$\not\models$	$\leftarrow Q(a,c)$
(1)	$I_0(\neg R(a))$	$\not\models$	$R(a) \leftarrow$
			unsatisfiable

IMs - Classification

Recall:

- Two-level calculi: instance generation separated from SAT solving – may use any SAT solver
- One-level calculi: monolithic, with two modes of operation: First-order mode and propositional mode
  Developed so far:

IM	Extended Calculus
DC	Connection Method, Tableaux
DCTP	Tableaux
OSHT	Hyper Tableaux
Hyper Tableaux NG	Hyper Tableaux
FDPLL	DPLL
ME	DPLL

IMs - Classification

Recall:

- Two-level calculi: instance generation separated from SAT solving – may use any SAT solver
- One-level calculi: monolithic, with two modes of operation: First-order mode and propositional mode
  Developed so far:

IM	Extended Calculus
DC	Connection Method, Tableaux
DCTP	Tableaux
OSHT	Hyper Tableaux
Hyper Tableaux NG	Hyper Tableaux
FDPLL	DPLL
ME	DPLL

Next: one-level calculus: FDPLL (simpler) / ME (better)

Motivation for FDPLL/ME

FDPLL: lifting of propositional core of DPLL to ***F***irst-order logic

Why? [Baumgartner, 2000]

- Lift very efficient propositional DPLL techniques to first-order
- From propositional DPLL: binary splitting, backjumping, learning, restarts, selection heuristics, simplification, ...
  Not all achieved yet; simplification not in FDPLL, but in ME
- Successful first-order techniques: unification, special treatment of unit clauses, subsumption (limited)
- For *theorem proving:* alternative to established methods
- For *model computation:*
  counterexamples, diagnosis, abduction, planning, nonmonotonic reasoning,... – largely unexplored

Contents FDPLL/ME Part

- Propositional DPLL as a semantic tree method
- FDPLL calculus
- Model Evolution calculus
- FDPLL/ME vs. OSHL
- FDPLL/ME vs. Inst-Gen

Propositional DPLL as a Semantic Tree Method

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

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

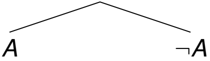
⟨**empty tree**⟩

- A Branch stands for an interpretation
- *Purpose of splitting:* satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

Propositional DPLL as a Semantic Tree Method

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

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

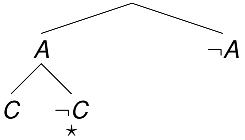


- A Branch stands for an interpretation
- *Purpose of splitting:* satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

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) $\neg D \vee \neg B$

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

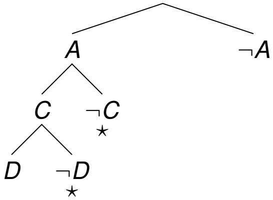


- A Branch stands for an interpretation
- *Purpose of splitting:* satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

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) $\neg D \vee \neg B$

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



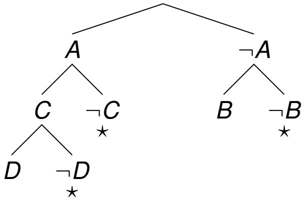
Model $\{A, C, D\}$ **found.**

- A Branch stands for an interpretation
- *Purpose of splitting:* satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by 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) $\neg D \vee \neg B$

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

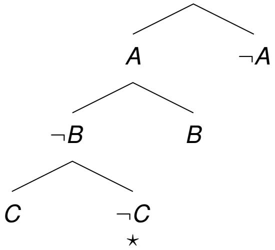


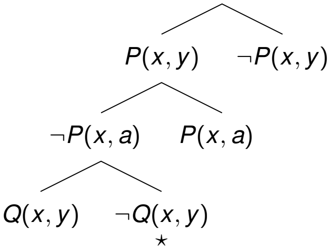
Model $\{B\}$ found.

- A Branch stands for an interpretation
- *Purpose of splitting:* satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

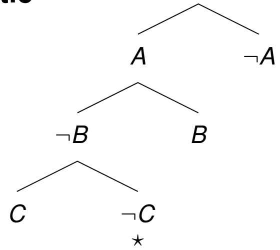
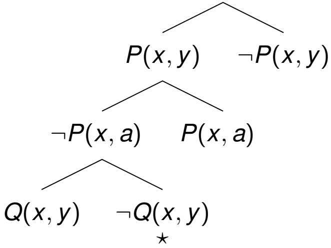
Meta-Level Strategy: Lifted data structures

	DPLL	**FDPLL**
Clauses	$B \vee C$	$P(x,y) \vee Q(x,x)$

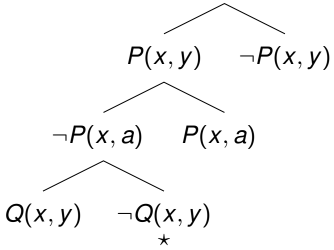




Meta-Level Strategy: Lifted data structures

	DPLL	**FDPLL**
Clauses	$B \vee C$	$P(x,y) \vee Q(x,x)$
Semantic Trees		

First-Order Semantic Trees



Issues:

- How are variables treated?
  (a) Universal?, (b) Rigid?, (c) Schematic!
- What is the interpretation represented by a branch?
  Clue to understanding of FDPLL (as is for Inst-Gen)

Extracting an Interpretation from a Branch

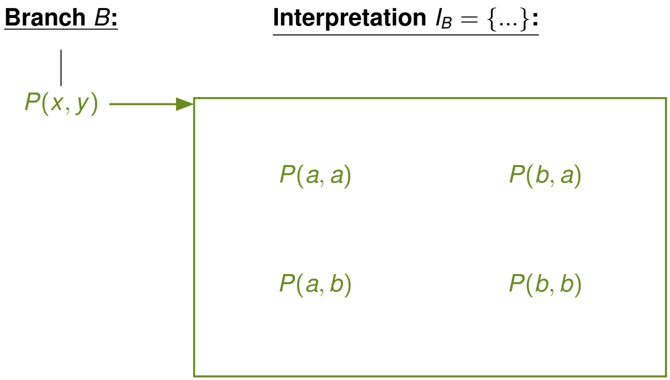
Branch B:

$P(x, y)$

Interpretation $I_B = \{...\}$:

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

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 the opposite truth value

Extracting an Interpretation from a Branch

Branch B:

$P(x, y)$

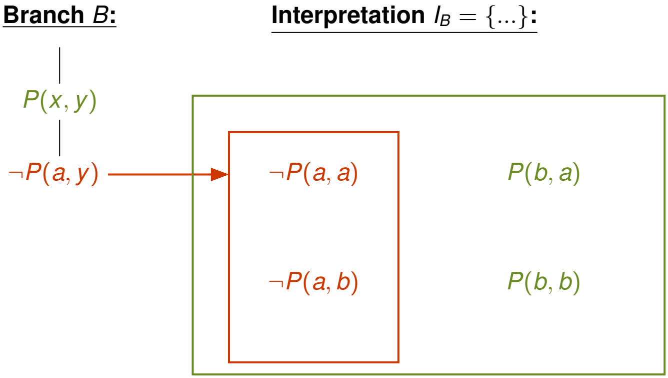
$\neg P(a, y)$

Interpretation $I_B = \{...\}$:

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

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

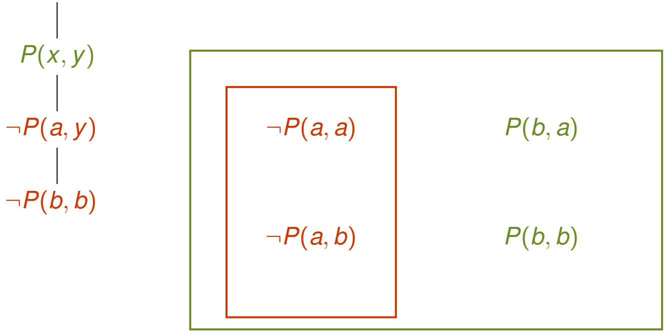
Extracting an Interpretation from a Branch



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

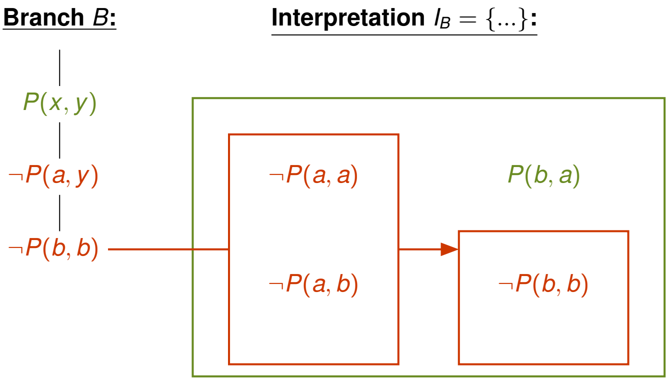
Extracting an Interpretation from a Branch

Branch B: **Interpretation $I_B = \{...\}$:**



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

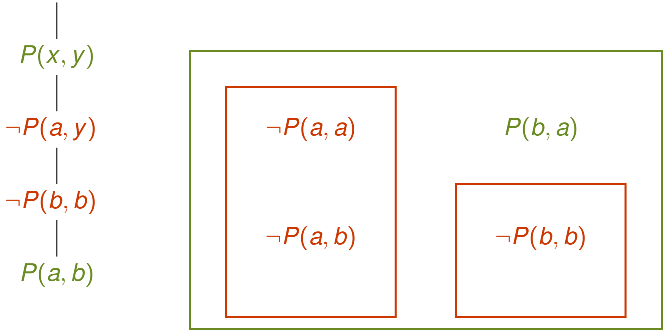
Extracting an Interpretation from a Branch



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

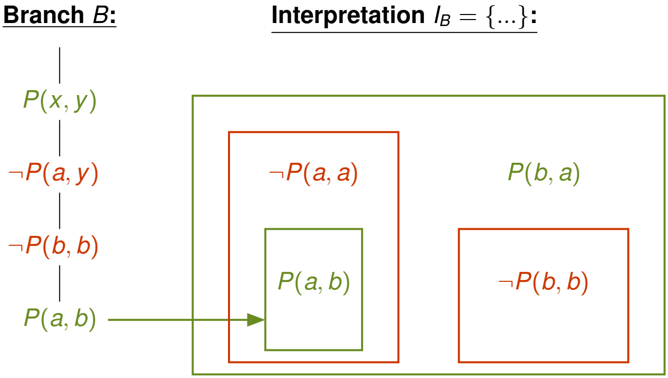
Extracting an Interpretation from a Branch

Branch B: **Interpretation $I_B = \{...\}$:**



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

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 the opposite truth value

Extracting an Interpretation from a Branch

Branch B:

$P(x,y)$

$\neg P(a,y)$

$\neg P(b,b)$

$P(a,b)$

Interpretation $I_B = \{\ldots\}$:

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

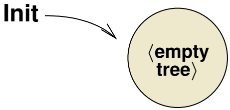
- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying the opposite truth value
- The order of literals does not matter

FDPLL Calculus - Main Loop

Input: a clause set S

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

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

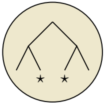


FDPLL Calculus - Main Loop

Input: a clause set S

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

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

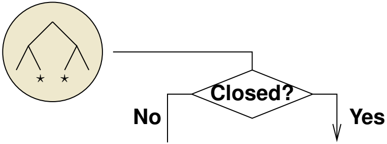


FDPLL Calculus - Main Loop

Input: a clause set S

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

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

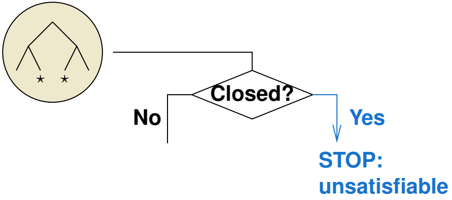


FDPLL Calculus - Main Loop

Input: a clause set S

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

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

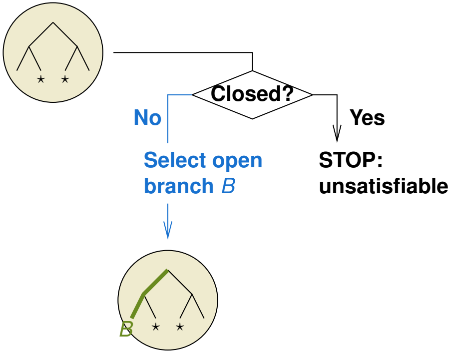


FDPLL Calculus - Main Loop

Input: a clause set S

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

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

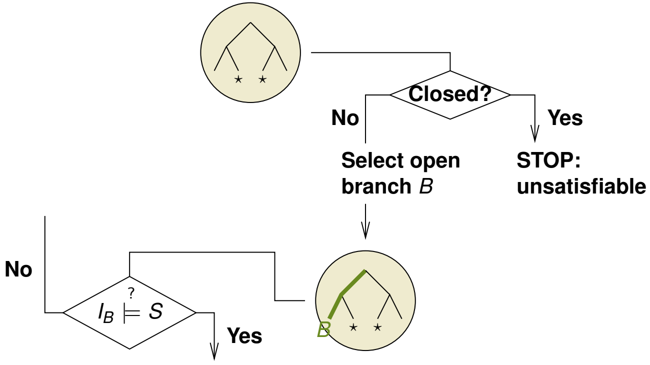


FDPLL Calculus - Main Loop

Input: a clause set S

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

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

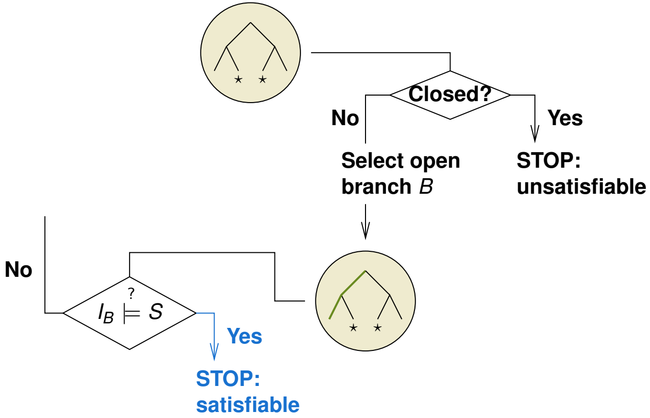


FDPLL Calculus - Main Loop

Input: a clause set S

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

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

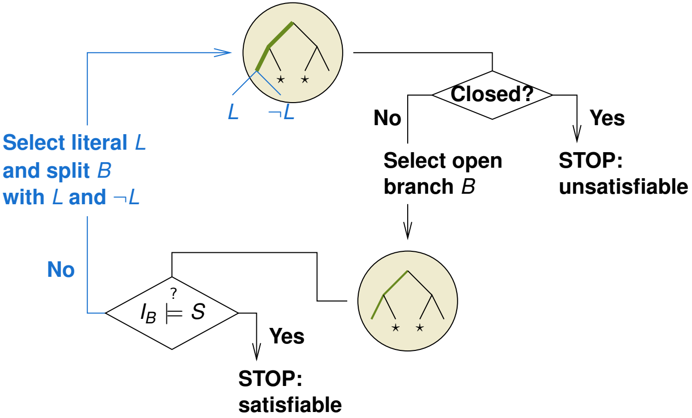


FDPLL Calculus - Main Loop

Input: a clause set S

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

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

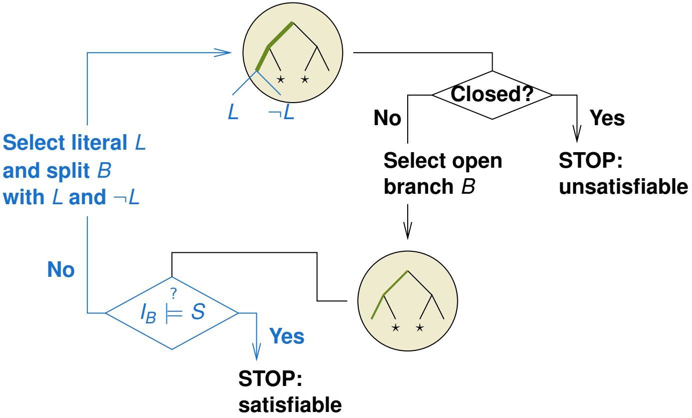


FDPLL Calculus - Main Loop

Input: a clause set S

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

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



Not here: FDPLL derivation rules for testing $I_B \models S$ and Splitting

FDPLL – Model Computation Example

Computed Model (as output by Darwin implementation)

```
(1)  train(X,Y) ; flight(X,Y).      %% train from X to Y or flight..
(2)  -flight(sb,X).                 %% no flight from sb 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
                     connect(Y,Z)   %% relation
```

FDPLL – Model Computation Example

Computed Model (as output by Darwin implementation)

```
(1)  train(X,Y) ; flight(X,Y).       %% train from X to Y or flight..
(2)  -flight(sb,X).                  %% no flight from sb 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
                     connect(Y,Z)    %% relation
```

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

FDPLL Model Computation Example - Derivation

•

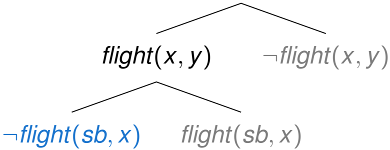
Clause instance used in inference: $\mathit{train}(x, y) \lor \mathit{flight}(x, y)$

FDPLL Model Computation Example - Derivation

$flight(x, y)$ $\neg flight(x, y)$

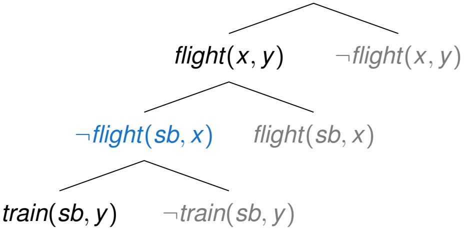
Clause instance used in inference: $\neg flight(sb, x)$

FDPLL Model Computation Example - Derivation



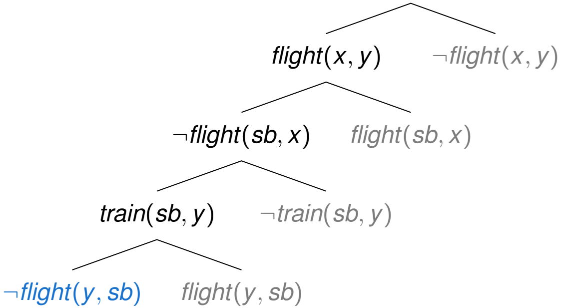
Clause instance used in inference: $\mathit{train}(sb, y) \vee \mathit{flight}(sb, y)$

FDPLL Model Computation Example - Derivation



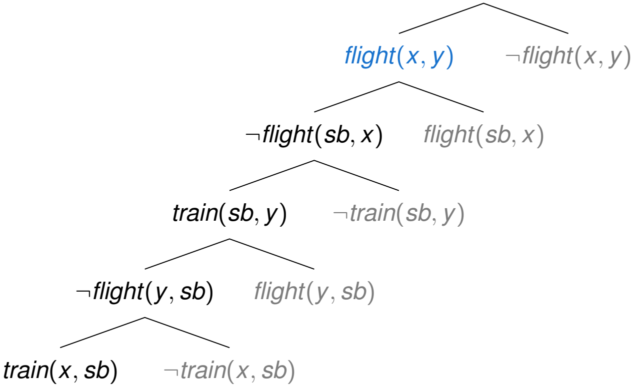
Clause instance used in inference: $flight(sb, y) \vee \neg flight(y, sb)$

FDPLL Model Computation Example - Derivation



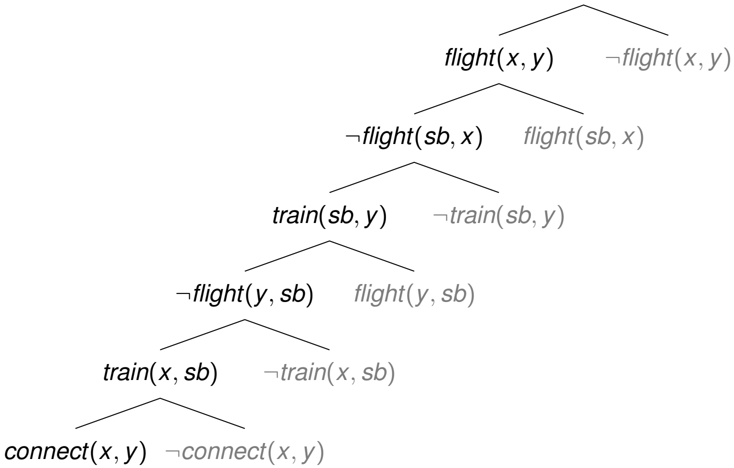
Clause instance used in inference: $train(x, sb) \lor flight(x, sb)$

FDPLL Model Computation Example - Derivation



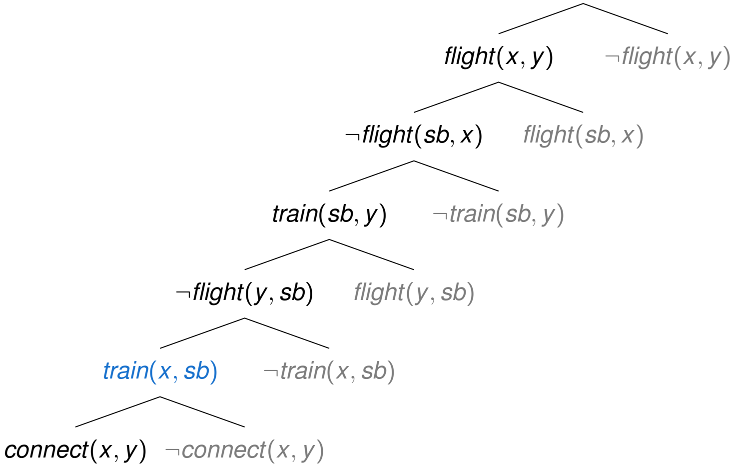
Clause instance used in inference: $connect(x, y) \lor \neg flight(x, y)$

FDPLL Model Computation Example - Derivation



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

FDPLL Model Computation Example - Derivation



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

Model Evolution (ME) Calculus

- Same motivation as for FDPLL: lift propositional DPLL to first-order
- Loosely based on FDPLL, but not really an “extension”
- Extension of Tinelli’s sequent-style DPLL [Tinelli, 2002]
- See [Baumgartner and Tinelli, 2003] for calculus, [**?**] for implementation “Darwin”

Difference to FDPLL

- Systematic treatment of universal and schematic variables
- Includes first-order versions of unit simplification rules
- Presentation as a sequent-style calculus, to cope with dynamically changing branches and clause sets due to simplification

FDPLL/ME vs. OSHL

Recall OSHL:

- Incrementally modify I_0
  Modified interpretation represented as $I_0(L_1, \ldots, L_m)$
- Find next ground instance $C\gamma$ by unifying subclause of C against $(L_1, \ldots, L_m)$ and guess Herbrand-instantiation of rest clause, so that $I_0(L_1, \ldots, L_m) \not\models C\gamma$

FDPLL/ME

- Initial interpretation I_0 is a trival one (e.g. "false everywhere")
- But $(L_1, \ldots, L_m)$ is a set of first-order literals now
- Find next (possibly) non-ground instance $C\sigma$ by unifying C against $(L_1, \ldots, L_m)$ so that $(L_1, \ldots, L_m) \not\models C\sigma$

FDPLL/ME vs. Inst-Gen

FDPLL/ME and Inst-Gen temporarily switch to propositional reasoning. But:

Inst-Gen (and other two-level calculi)

- Use the $\bot$-version $S_\bot$ of the current clause set S

$\Rightarrow$ Works globally on clause sets

- Flexible: may switch focus all the time – but memory problem (?)

FDPLL/ME (and other one-level calculi)

- Use the \$-version of the current branch

$\Rightarrow$ Works locally in context of current branch

- Not so flexible – but don't expect memory problems:
  FDPLL/ME need not keep *any* clause instance
  DCTP needs to keep clause instances only along current branch

Applicability/Non-Applicability of IMs

- Comparison: Resolution vs. Tableaux vs. IMs
- Conclusions from that

Resolution vs. Tableaux vs. IMs

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

Resolution

- Resolution may generate clauses of unbounded length:

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

\- Does not decide function-free clause sets

\- Complicated to extract model

\+ (Ordered) Resolution very good on some classes, Equality

Resolution vs. Tableaux vs. IMs

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

Rigid Variables Approaches (Tableaux, Connection Methods)

- Have to use unbounded number of variants per clause:

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

  - Weak redundancy criteria
  - Difficult to exploit proof confluence
    Usual calculi backtrack more than theoretically necessary
    But see [Giese, 2001], [Baumgartner *et al.*, 1999], [Beckert, 2003]
- Model Elimination: *goal-orientedness* compensates drawback

Difficulty with Rigid Variable Methods

Rigid variable methods "destructively" modify data structure

S: $\forall x(P(x) \lor Q(x))$
$\neg P(a)$
$\neg P(b)$
$\neg Q(b)$

(1) $P(X) \lor Q(X)$

(2) $P(X) \lor Q(X)$
$\neg P(a)$

(3) $P(a) \lor Q(a)$
$\neg P(a)$

(5) $P(a) \lor Q(a)$
$\neg P(a)$
$P(X') \lor Q(X')$
$\neg P(b)$

(7) $P(a) \lor Q(a)$
$\neg P(a)$
$P(b) \lor Q(b)$
$\neg P(b)$
$\neg Q(b)$

- Connection method (and tableaux) proof confluent: no deadends
- Difficulty to find fairness criterion due to "destructive" nature
- All IMs are non-destructive – no problem here

Resolution vs. Tableaux vs. IMs

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

Instance Based Methods

- May need to generate and keep *proper* instances of clauses:

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

- Cannot use subsumption: weaker than Resolution
- Clauses do not grow in length, no recombination of clauses: better than Resolution, same as in rigid variables approaches

\+ Need not keep variants: better than rigid variables approaches

Applicability/Non-Applicability of IMs: Conclusions

Suggested applicability for IMs:

- Near propositional clause sets
- Clause sets without function symbols (except constants)
  E.g. Translation from basic modal logics, Datalog
- Model computation (sometimes)

Other methods (currently?) better at:

- Goal orientation
- Equality, theory reasoning
- Many decidable fragments (Guarded fragment, two-variable fragment)

Part II: A Closer Look

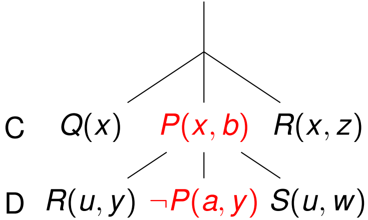
- Disconnection calculus
- Theory Reasoning and Equality
- Implementations and Techniques
  - Available Implementations
  - Proof Procedures
  - Exploiting SAT techniques

Disconnection Tableaux

The Disconnection Calculus(I)

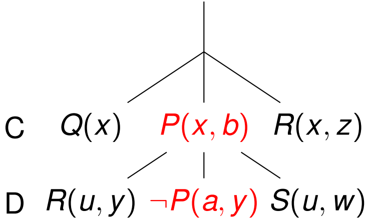
- Analytic tableau calculus for first order clause logic
- Introduced by J.-P. Billon (1996)
- Special characteristics of calculus:
  - No *rigid* variables
  - No *variants* in tableau
  - *Proof confluence*: One proof tree only, no backtracking in search
  - *Saturated* branches as indicator of satisfiability
  - *Decision procedure* for certain classes of formulae
- Related methods: hyper linking, hyper tableaux, first order Davis-Putnam ...

The Disconnection Calculus (II): Proof Rule Linking



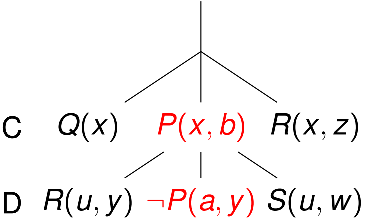
potentially complementary literals on path

The Disconnection Calculus (II): Proof Rule Linking

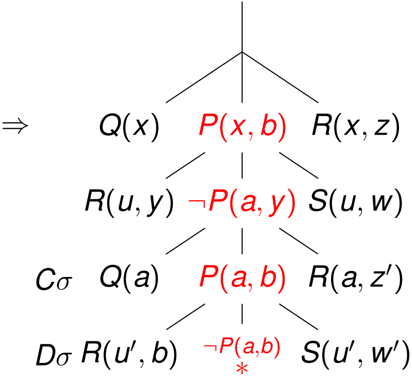


unifier for literals:
$\{x/a, y/b\}$

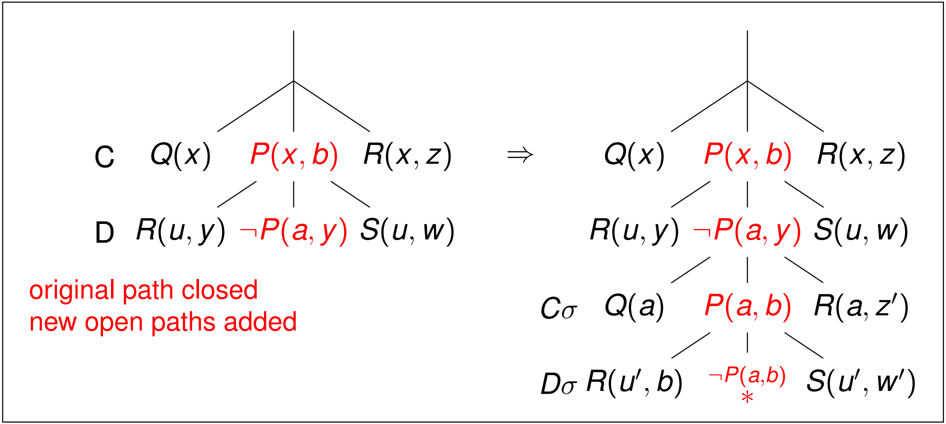
The Disconnection Calculus (II): Proof Rule Linking



append instances with substitution $\{x/a, y/b\}$ to path



The Disconnection Calculus (II): Proof Rule Linking



- Concept of $\forall$-closure of branches
  closure by simultaneous instantiation of all variables by the same constant: path with $P(x,y)$ and $\neg P(z,z)$ is closed

Proof Search in the Disconnection Calculus

- Proof process in two phases:
  - An initial active path through the formula is don't-care nondeterministically selected
  - Using the links contained in the active path, instances of linked clauses are used to build a tableau
- An open tableau path may be selected don't-care nondeterministically, it becomes the next active path
- Each link can be used only once on a path (explains the name "disconnection")
- Absence of usable links (saturation of a path) indicates satisfiability of the formula
- Only requirement for (strong) completeness: fairness of link selection

An Example Proof

Input Clauses $P(x,z) \lor \neg P(x,y) \lor \neg P(y,z)$

$$P(b,c)$$

$$P(a,b)$$

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

An Example Proof

Input Clauses $P(x,z) \lor \neg P(x,y) \lor \neg P(y,z)$

$P(b,c)$

$P(a,b)$

$\neg P(a,c)$

An Example Proof

Input Clauses $P(x,z) \lor \neg P(x,y) \lor \neg P(y,z)$

$P(b,c)$

$P(a,b)$

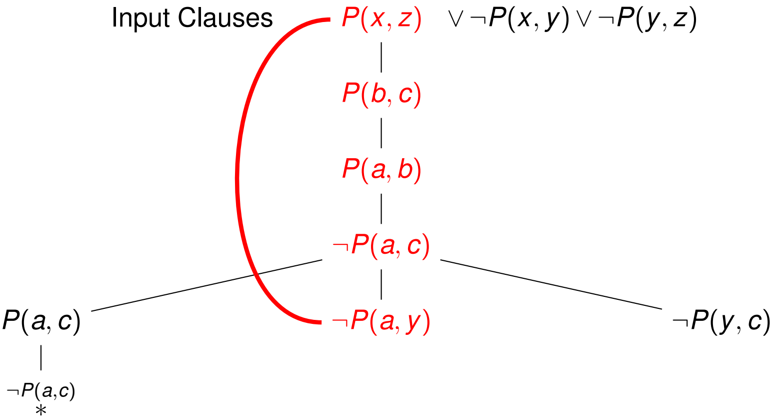
$\neg P(a,c)$

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

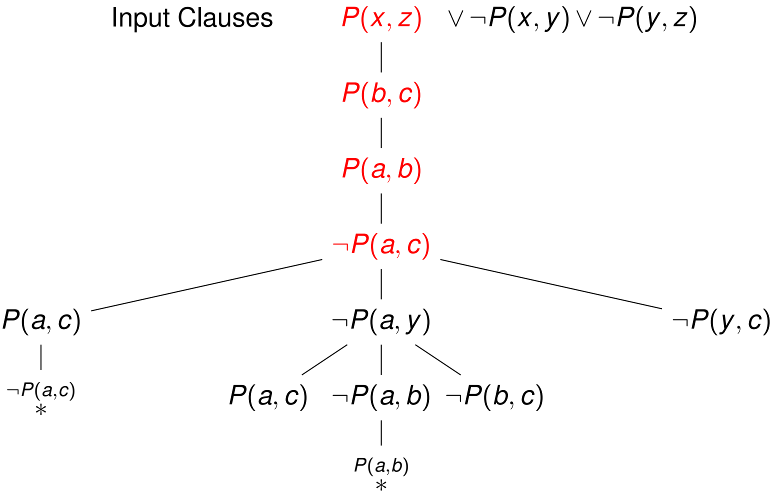
$\neg P(a,c)$

$*$

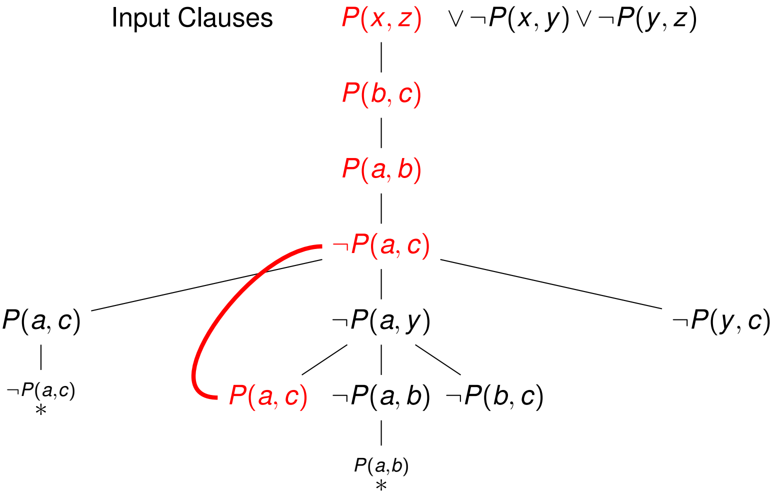
An Example Proof



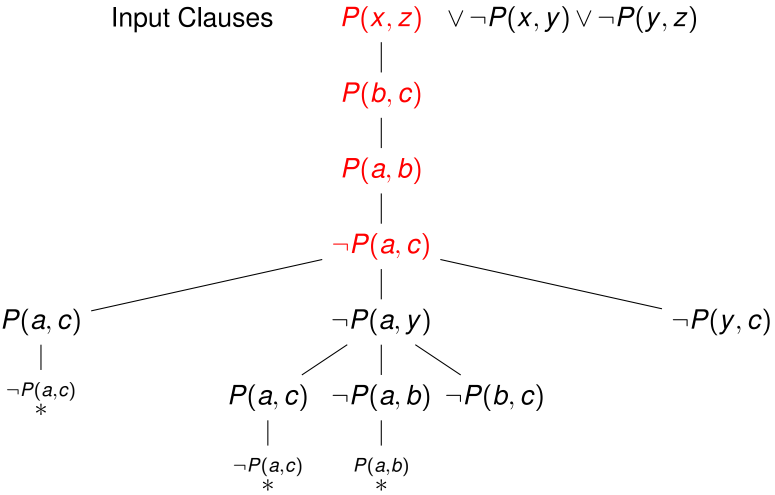
An Example Proof



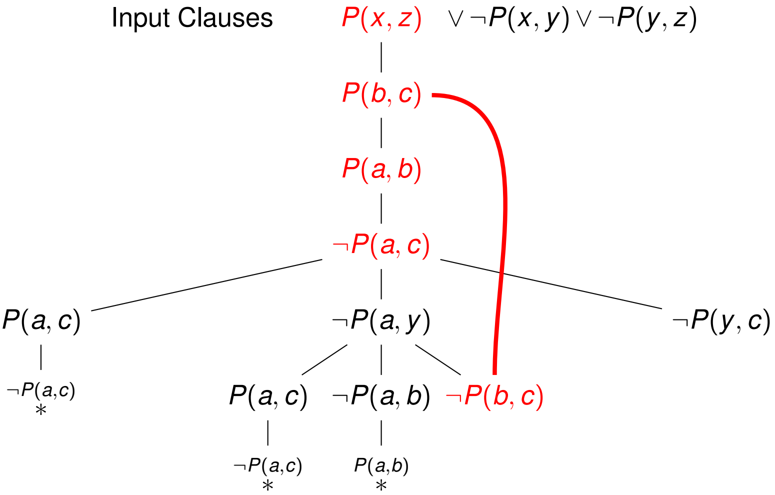
An Example Proof



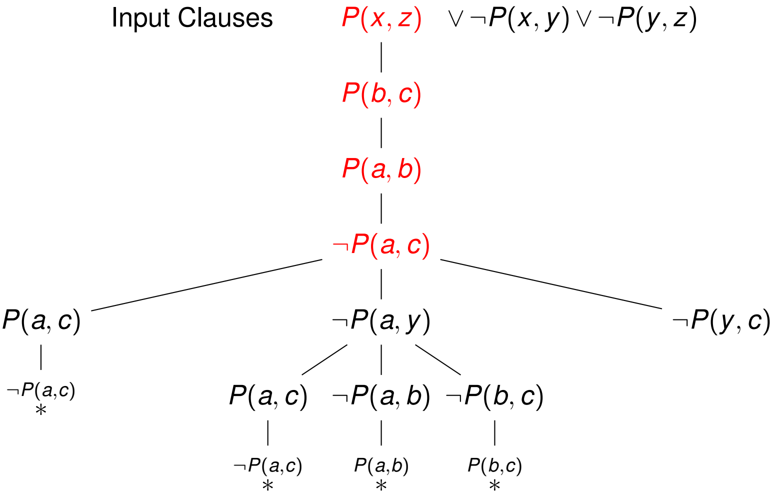
An Example Proof



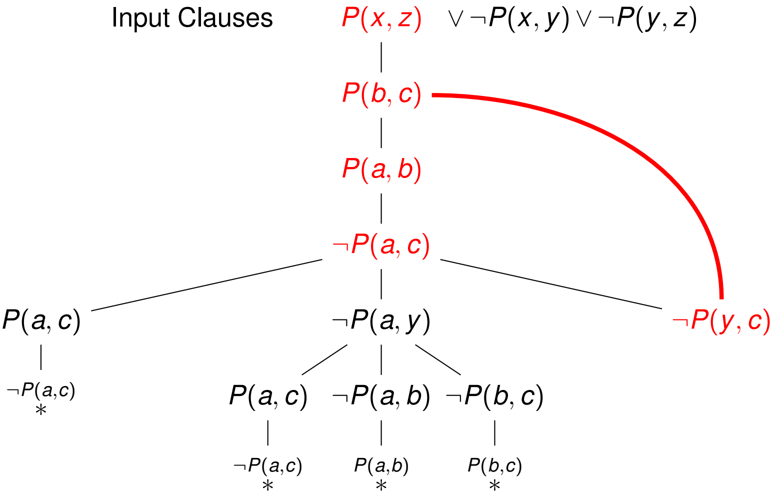
An Example Proof



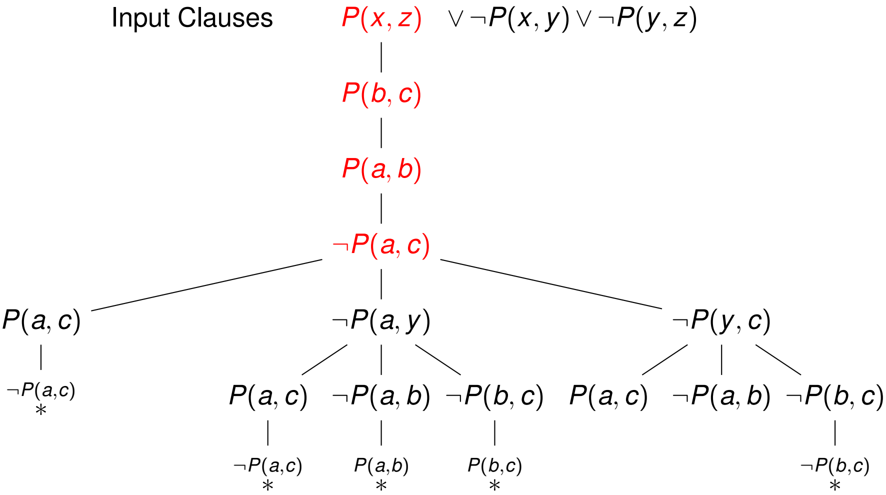
An Example Proof



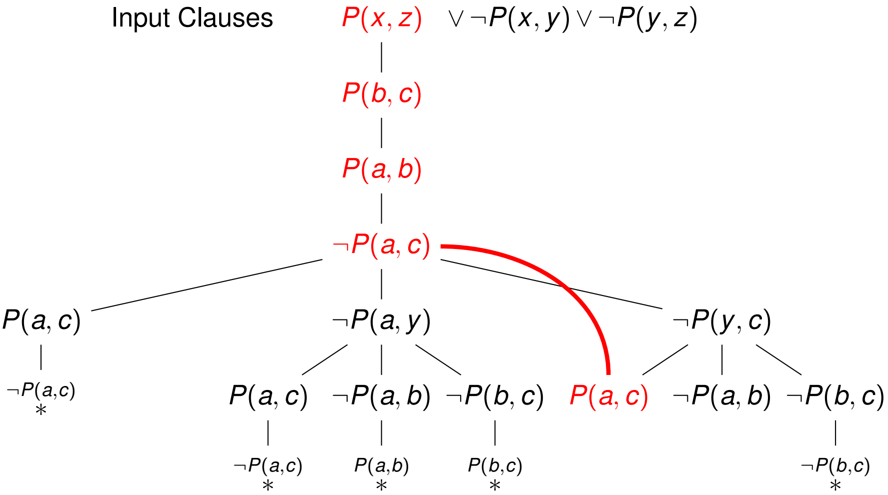
An Example Proof



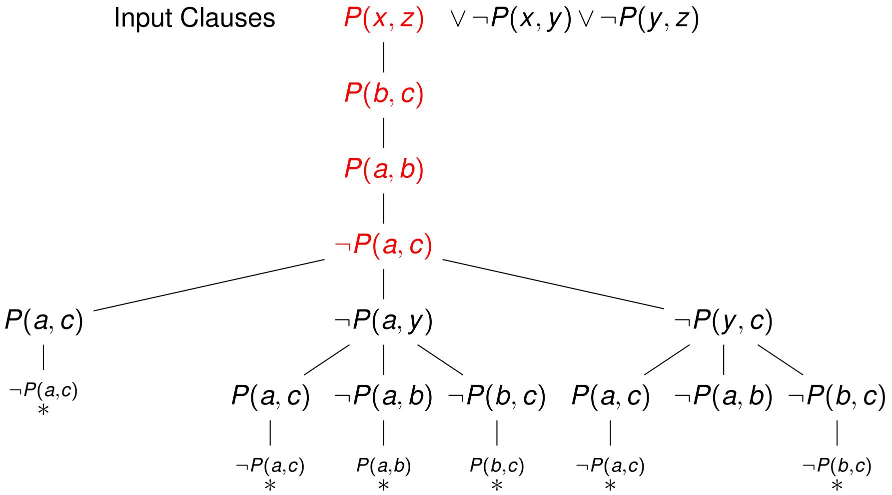
An Example Proof



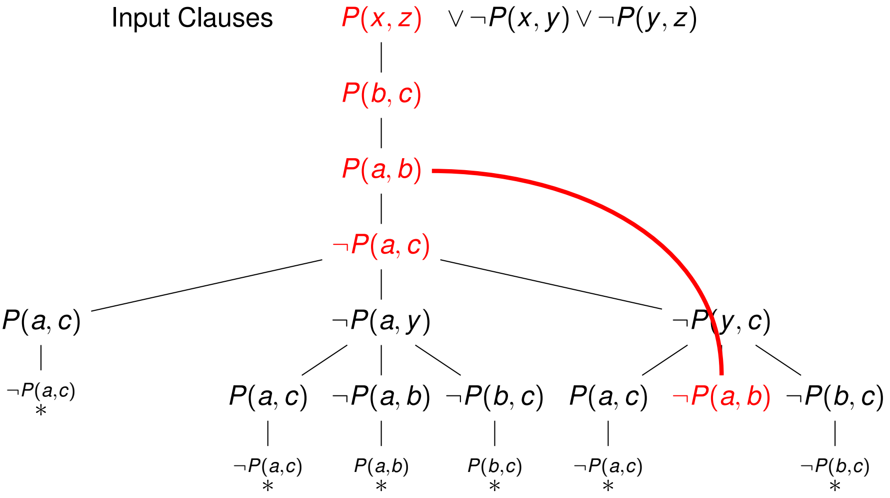
An Example Proof



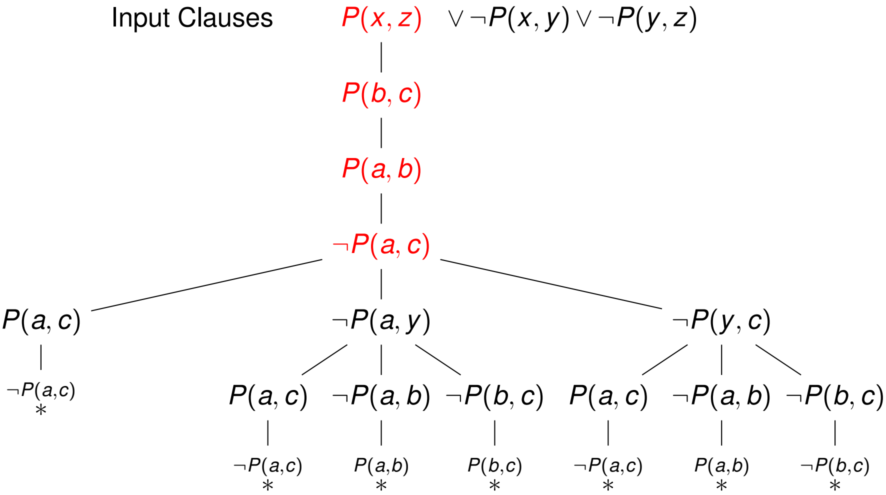
An Example Proof



An Example Proof



An Example Proof



Variant Freeness

- Two clauses are *variants* if they can be obtained from each other by variable renaming
- A tableau is *variant-free* if no branch contains literals l and k where the clauses of l and k are variants
- All disconnection tableaux are required to be variant-free
- Variant-freeness provides essential pruning (weak form of subsumption)
- Vital for model generation
- Implies the idea of *branch saturation*:
  A branch is *saturated* if it cannot be extended in a variant-free manner

Failed Proof Attempts

- Proof attempts may fail - what happens then?

Failed Proof Attempts

- Proof attempts may fail - what happens then?
- In order to show this,
  we will change one clause in the previous example: the signs are inverted

Input Clauses $\neg P(x,z) \quad \vee P(x,y) \vee P(y,z)$

$$\neg P(x,z)$$
$$|$$
$$P(b,c)$$
$$|$$
$$P(a,b)$$
$$|$$
$$\neg P(a,c)$$

Failed Proof Attempts

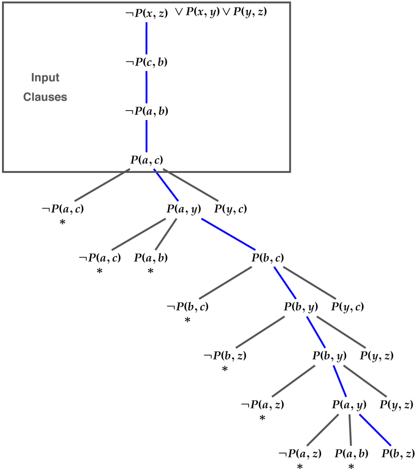
- Proof attempts may fail - what happens then?
- In order to show this,
  we will change one clause in the previous example: the signs are inverted

Input Clauses $\neg P(x,z) \quad \vee P(x,y) \vee P(y,z)$

$$\begin{array}{c} \neg P(x,z) \\ | \\ P(b,c) \\ | \\ P(a,b) \\ | \\ \neg P(a,c) \end{array}$$

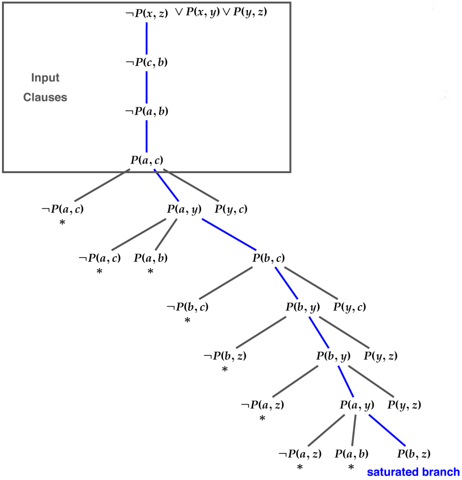
- Again, we attempt to find a proof

A Saturated Open Tableau



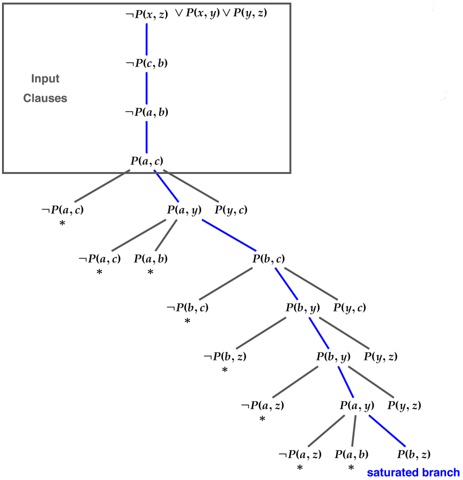
- This open tableau cannot be closed

A Saturated Open Tableau



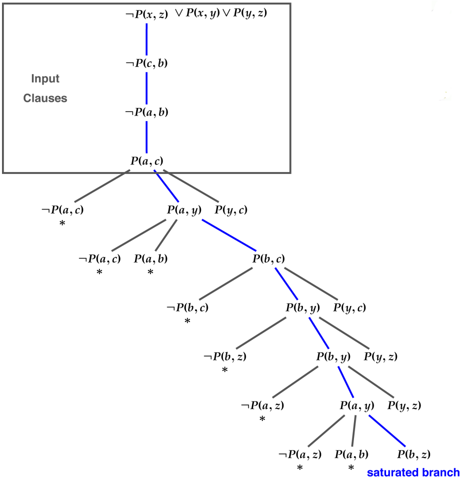
- This open tableau cannot be closed
- Indicated branch is saturated

A Saturated Open Tableau



- This open tableau cannot be closed
- Indicated branch is saturated
- Saturated open branch provides model

A Saturated Open Tableau



- This open tableau cannot be closed
- Indicated branch is saturated
- Saturated open branch provides model
- How to extract model?

- *Instance Preserving Enumerations*: lists of literal occurrences on a path
- Path literals are partially ordered in enumeration (not unique)
- Each literal must occur before all more general instances of itself
- Instance preserving enumeration of a saturated open branch implies model
- Example: For the open (sub-) branch

$\neg P(a)$

$P(x)$

$\neg P(c)$

With Herbrand universe $\{a, b, c, d, e\}$ and enumeration

$$[\neg P(a) \quad \neg P(c) \quad P(x)]$$

the model implied is
$\{\neg P(a), P(b), \neg P(c), P(d), P(e)\}$

Model Extraction

We extract an instance preserving enumeration for the open branch of the preceding tableau:

$$\boxed{\neg P(c,b)}$$

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

$$\boxed{P(a,c)}$$

$$\neg P(a,c) \lor P(a,b) \lor \boxed{P(b,c)}$$

$$\neg P(a,z) \lor P(a,b) \lor \boxed{P(b,z)}$$

$$\neg P(a,c) \lor \boxed{P(a,y)} \lor P(y,c)$$

$$\neg P(b,c) \lor \boxed{P(b,y)} \lor P(y,c)$$

$$\neg P(b,z) \lor \boxed{P(b,y)} \lor P(y,z)$$

$$\neg P(a,z) \lor \boxed{P(a,y)} \lor P(y,z)$$

$$\boxed{\neg P(x,z)} \lor P(x,y) \lor P(y,z)$$

Model Extraction

We extract an instance preserving enumeration for the open branch of the preceding tableau:

$\neg P(c,b)$

$\neg P(a,b)$

$P(a,c)$

$\neg P(a,c) \lor P(a,b) \lor P(b,c)$

$\neg P(a,z) \lor P(a,b) \lor P(b,z)$

$\neg P(a,c) \lor P(a,y) \lor P(y,c)$

$\neg P(b,c) \lor P(b,y) \lor P(y,c)$

$\neg P(b,z) \lor P(b,y) \lor P(y,z)$

$\neg P(a,z) \lor P(a,y) \lor P(y,z)$

$\neg P(x,z) \lor P(x,y) \lor P(y,z)$

From which we get the finite Herbrand model:

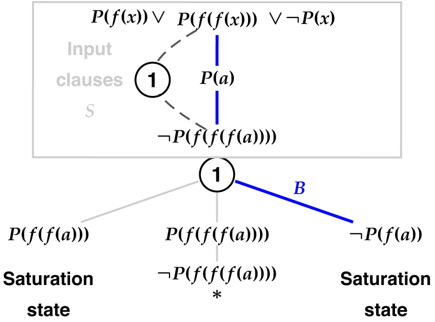
$$\{ \quad \neg P(c,b), \neg P(a,b), P(a,c), \quad P(b,c), P(b,a), P(b,b), \quad P(a,a), \neg P(c,a), \neg P(c,c) \quad \}$$

Infinite Herbrand Models

Model extraction also works for infinite Herbrand universes

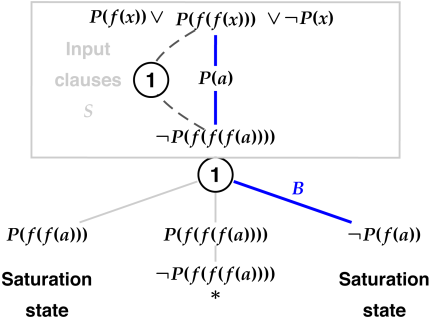
Infinite Herbrand Models

Model extraction also works for infinite Herbrand universes
Given a saturated tableau with open branch B:



Infinite Herbrand Models

Model extraction also works for infinite Herbrand universes
Given a saturated tableau with open branch B:



The enumeration for B
$\neg P(f(f(f(a)))),\ \neg P(f(a)),\ P(a),\ P(f(f(x)))$
implies a finite representation of an

infinite Herbrand model:
$\{\neg P(f(f(f(a)))), \neg P(f(a)), P(a)\}, \{P(f(f(\mathfrak{s})))\}$
with the constraint $\mathfrak{s} \neq f(a)$, where $\mathfrak{s}$ ranges over the Herbrand universe of S.

Completeness

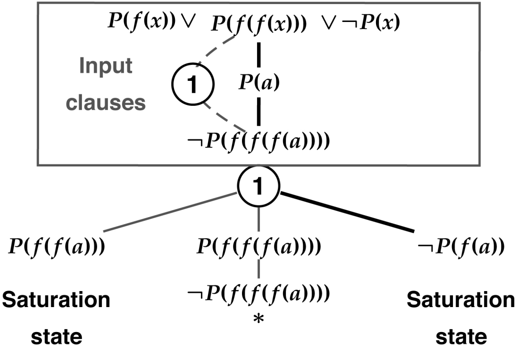
- Basic concept: open saturated branch represents partial model
- Non-equational case: branch determines path through Herbrand set

  non-ground open branch (non-rigid) ground Herbrand set

- Closed ground path corresponds to applicable link
  ⇔ contradicts saturation

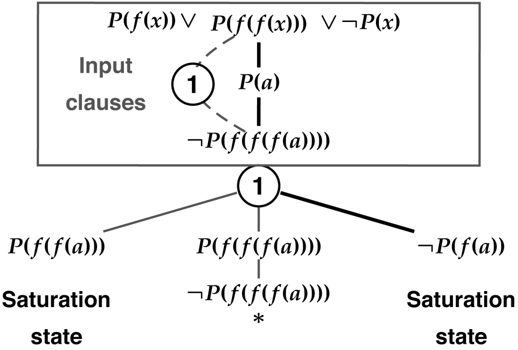
The Saturation Property

- Saturated open branch specifies a model (only such a branch)
- Model characterised as blue exception-based representation (EBR)



The Saturation Property

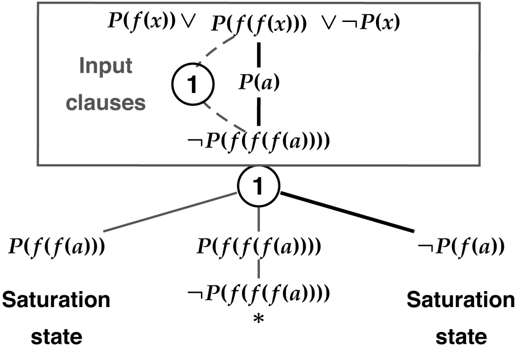
- Saturated open branch specifies a model (only such a branch)
- Model characterised as blue exception-based representation (EBR)



- Model: $\{\neg P(f(f(f(a)))), \neg P(f(a)), P(a)\} \cup \{P(f(f(\mathfrak{s}))) : \mathfrak{s} \neq f(a)\}$

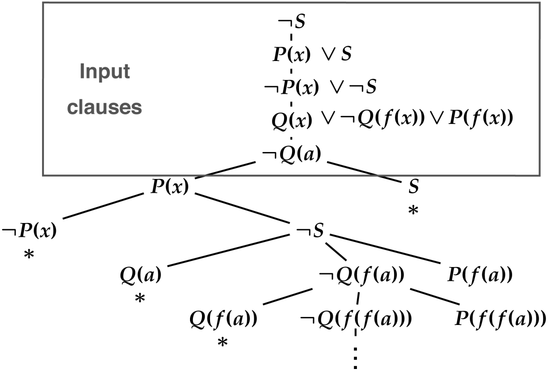
The Saturation Property

- Saturated open branch specifies a model (only such a branch)
- Model characterised as blue exception-based representation (EBR)



- EBR for model: $\{P(a), \neg P(f(a)), P(f(f(x))), \neg P(f(f(f(a))))\}$

An Example for Non-Termination



- The above problem is obviously satisfiable (P true, S and Q false)
- However, in general, the disconnection calculus does not terminate
- Termination fragile, depends on branch selection function

The Problem

- Here, the model is approximated, but not finitely represented
  $\{P(x), \neg S, \neg Q(a), \neg Q(f(a)), \neg Q(f(f(a))), \neg Q(f(f(f(a)))) \ldots\}$
- Observation: linking instances are subsumed by path literal $P(x)$
- But: general subsumption does not work
- What can we do?

Link Blocking

- Original idea of model characterisation:
  - Currently considered branch is seen as an interpretation I
  - If a literal L is on branch, all instances of L are considered true in I
  - if a conflict occurs (a link is on the branch), the link is applied and I is modified

Link Blocking

- Original idea of model characterisation:
  - Currently considered branch is seen as an interpretation I
  - If a literal L is on branch, all instances of L are considered true in I
  - if a conflict occurs (a link is on the branch), the link is applied and I is modified
- Consequence: Ignore clauses subsumed by I
- Concept of temporary link blocking
  - Path subgoal L will disable all links producing literals $K = L\sigma$
  - Unblocking of links occurs when a conflict involving L is resolved, i.e. the interpretation I is changed

Link Blocking

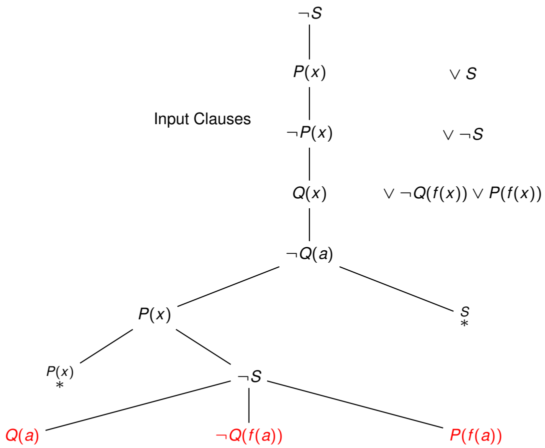
- Original idea of model characterisation:
  - Currently considered branch is seen as an interpretation I
  - If a literal L is on branch, all instances of L are considered true in I
  - if a conflict occurs (a link is on the branch), the link is applied and I is modified
- Consequence: Ignore clauses subsumed by I
- Concept of temporary link blocking
  - Path subgoal L will disable all links producing literals $K = L\sigma$
  - Unblocking of links occurs when a conflict involving L is resolved, i.e. the interpretation I is changed
- Similar to productivity restriction in ME

Candidate Models

- Precise criteria needed to find out whether a literal is blocking
- EBRs are lists of branch literals partially sorted according to respective specialisation
- Candidate model (CM): EBR enhanced by link blockings
- Blockings require a modified ordering on CMs, not necessarily based on instantiation
- Interpretation of a literal L given by CM-matcher:
  the rightmost literal in CM subsuming L or $\sim L$

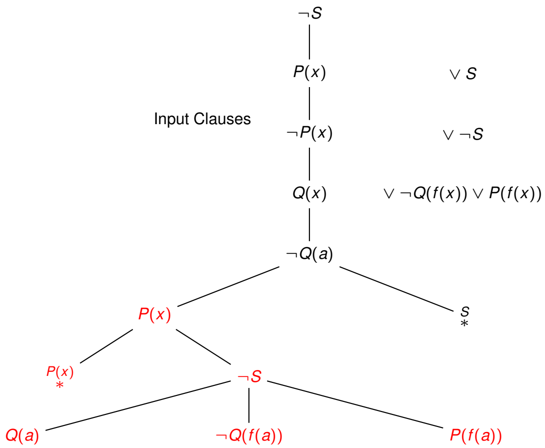
Link Blocking Example

- The non-termination example revisited



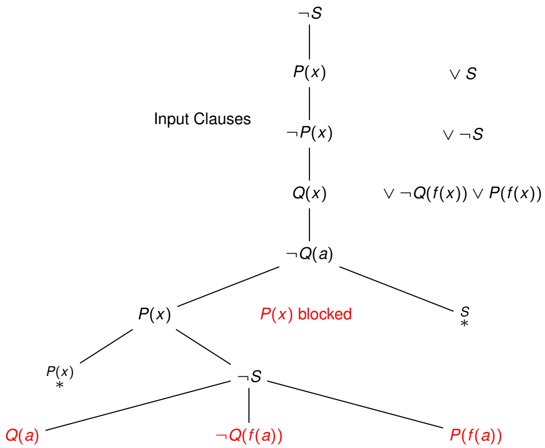
Link Blocking Example

- The non-termination example revisited



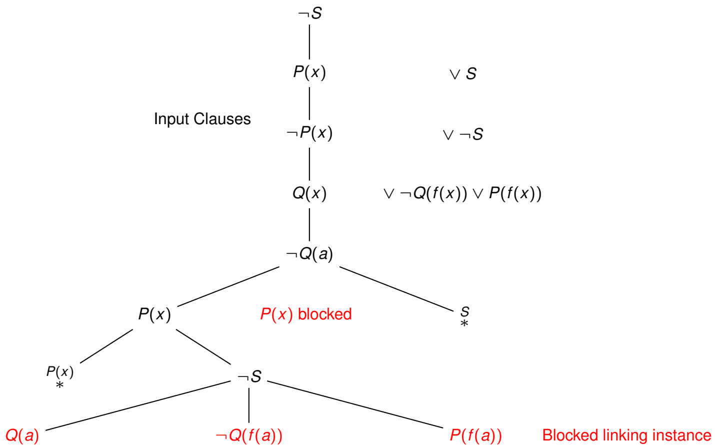
Link Blocking Example

- The non-termination example revisited



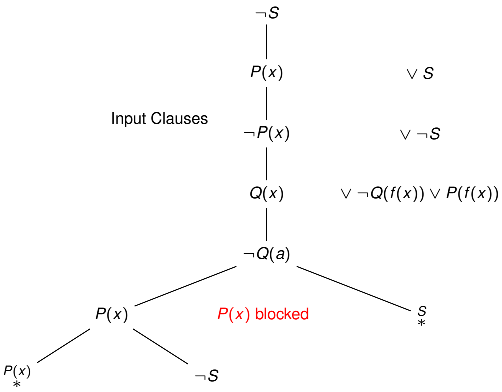
Link Blocking Example

- The non-termination example revisited



Link Blocking Example

- The non-termination example revisited



Saturation state

- Use of link blocking allows termination
- Largely independent of selection functions

Cyclic Link Blocking

$$Q(a,y) \lor \neg P(a,y)$$

$$\neg Q(x,b) \lor P(x,b)$$

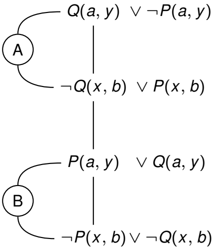
Unsatisfiable clause set

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

$$\neg P(x,b) \lor \neg Q(x,b)$$

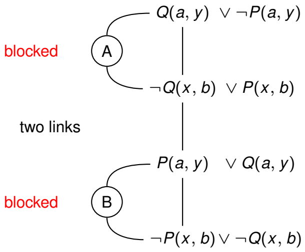
Cyclic Link Blocking

two links



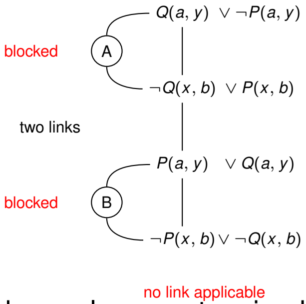
Unsatisfiable clause set

Cyclic Link Blocking



Unsatisfiable clause set

Cyclic Link Blocking

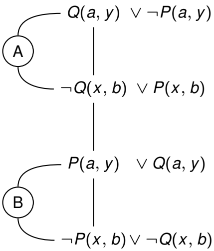


Unsatisfiable clause set

- For the above clause set, using blockings no refutation can be found
- Reason: The blocking relation for the clause set is cyclic
- To preserve completeness, blocking cycles must be avoided
- Well-founded ordering imposed on link blockings based on branch position

Cyclic Link Blocking Resolved

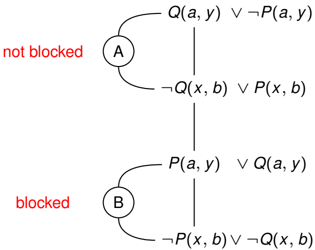
- We try again, this time with a blocking ordering



Unsatisfiable clause set

Cyclic Link Blocking Resolved

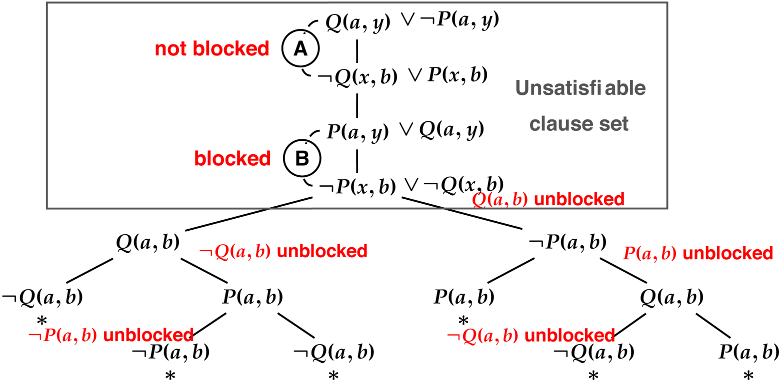
- We try again, this time with a blocking ordering



Unsatisfiable clause set

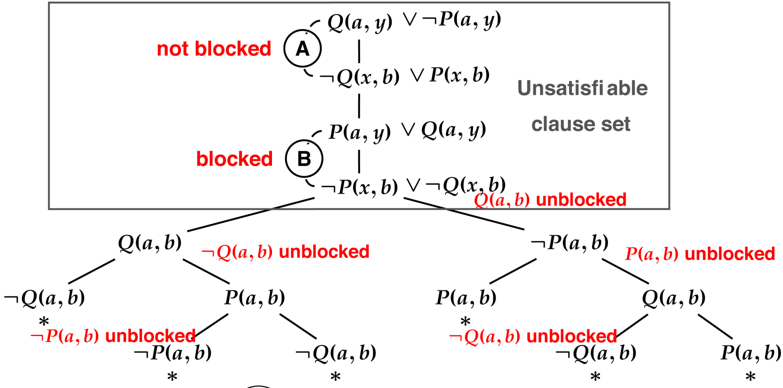
Cyclic Link Blocking Resolved

- We try again, this time with a blocking ordering



Cyclic Link Blocking Resolved

- We try again, this time with a blocking ordering



- Allowing link (A) to be applied, we initiate a series of blockings and unblockings that allow to refute the formula

The Basic Idea behind Completeness

- Completeness approach as in classical disconnection calculus:

$$\text{saturated open tableau branch } B^+ \\ \Longrightarrow \\ \text{consistent path } P^* \text{ through Herbrand set}$$

- P^* path literal in each ground clause is determined by CM-matcher
- Tricky part: There exists a matched literal in each ground clause
- Partial order of CM dynamically evolving with the branch
- Acyclicity of blocking relation ensures that partial order exists

FDPLL/ME vs. DCTP - Conceptual Difference

FDPLL/ME and DCTP use propositional version of current branch to determine branch closure. But:

DCTP

- Branch is closed if it contains both $L\perp$ and $\overline{L}\perp$ (two clauses involved)
- Inference rule guided syntactically: find connection among branch literals
- *n*-way branching on literals of clause instance $L_1 \vee \cdots \vee L_n$
  Can simulate FDPLL/ME binary branching to some degree (folding up)
- Need to keep clause instances along current branch

FDPLL/ME

- Branch is closed if \$-version falsifies some single clause
- Inference rule guided semantically: find falsified clause instance
- Binary branching on literals L - $\overline{L}$ taken from falsified clause instance
  Can simulate *n*-way branching clause literals in ground case
- Need not keep *any* clause instance, but better cache certain subclauses (remainders) to support heuristics

Peter Baumgartner and Cesare Tinelli.
The Model Evolution Calculus.
In Franz Baader, editor, *CADE-19 – The 19th International Conference on Automated Deduction*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 350–364. Springer, 2003.

Peter Baumgartner and Cesare Tinelli.
The model evolution calculus with equality.
In Robert Nieuwenhuis, editor, *CADE-20 – The 20th International Conference on Automated Deduction*, volume 3632 of *Lecture Notes in Artificial Intelligence*, pages 392–408. Springer, 2005.

Peter Baumgartner, Norbert Eisinger, and Ulrich Furbach.
A confluent connection calculus.
In Harald Ganzinger, editor, *CADE-16 – The 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 329–343, Trento, Italy, 1999. Springer.

Peter Baumgartner.

Hyper Tableaux — The Next Generation.
In Harry de Swaart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 60–76. Springer, 1998.

Peter Baumgartner.
FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure.
In David McAllester, editor, *CADE-17 – The 17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 200–219. Springer, 2000.

Bernhard Beckert.
Depth-first proof search without backtracking for free-variable clausal tableaux.
Journal of Symbolic Computation, 36:117–138, 2003.

Jean-Paul Billon.
The Disconnection Method.

In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, number 1071 in Lecture Notes in Artificial Intelligence, pages 110–126. Springer, 1996.

Alan Bundy, editor.
Automated Deduction — CADE 12, LNAI 814, Nancy, France, June 1994. Springer-Verlag.

T.J. Chinlund, M. Davis, P.G. Hinman, and M.D. McIlroy.
Theorem-Proving by Matching.
Technical report, Bell Laboratories, 1964.

Heng Chu and David A. Plaisted.
Semantically Guided First-Order Theorem Proving using Hyper-Linking.
In Bundy [1994], pages 192–206.

Koen Claessen and Niklas Sörensson.
New techniques that improve mace-style finite model building.

In Peter Baumgartner and Christian G. Fermüller, editors, *CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications*, 2003.

Martin Davis and Hilary Putnam.
A computing procedure for quantification theory.
Journal of the ACM, 7(3):201–215, July 1960.

M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem proving.
Communications of the ACM, 5(7), 1962.

Martin Davis, George Logemann, and Donald Loveland.
A machine program for theorem proving.
Communications of the ACM, 5(7):394–397, July 1962.

Martin Davis.
Eliminating the irrelevant from mechanical proofs.
In *Proceedings of Symposia in Applied Amthematics – Experimental Arithmetic, High Speed Computing and*

Mathematics, volume XV, pages 15–30. American Mathematical Society, 1963.

Harald Ganzinger and Konstantin Korovin.
New directions in instance-based theorem proving.
In *LICS - Logics in Computer Science*, 2003.

Martin Giese.
Incremental closure of free variable tableaux.
In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proc. Intl. Joint Conf. on Automated Reasoning IJCAR, Siena, Italy*, volume 2083 of *LNCS*, pages 545–560. Springer-Verlag, 2001.

Matthew L. Ginsberg and Andrew J. Parkes.
Satisfiability algorithms and finite quantification.
In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR'2000)*, pages 690–701. Morgan Kauffman, 2000.

J.N. Hooker, G. Rago, V. Chandru, and A. Shrivastava.
Partial Instantiation Methods for Inference in First Order Logic.
Journal of Automated Reasoning, 28(4):371–396, 2002.

Swen Jacobs and Uwe Waldmann.
Comparing Instance Generation Methods for Automated Reasoning.
In Bernhard Beckert, editor, *Proc. of TABLEAUX 2005*. Springer, 2005.

Shie-Jue Lee and David A. Plaisted.
Reasoning with Predicate Replacement, 1989.

S.-J. Lee and D. Plaisted.
Eliminating Duplicates with the Hyper-Linking Strategy.
Journal of Automated Reasoning, 9:25–42, 1992.

Reinhold Letz and Gernot Stenz.
Proof and Model Generation with Disconnection Tableaux.

In Robert Nieuwenhuis and Andrei Voronkov, editors, *LPAR*, volume 2250 of *Lecture Notes in Computer Science*. Springer, 2001.

William McCune.
A davis-putnam program and its application to finite first-order model search: Qusigroup existence problems.
Technical report, Argonne National Laboratory, 1994.

David A. Plaisted and Yunshan Zhu.
Ordered Semantic Hyper Linking.
In *Proceedings of Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 1997.

David A. Plaisted and Yunshan Zhu.
Ordered Semantic Hyper Linking.
Journal of Automated Reasoning, 25(3):167–217, 2000.

David Plaisted.
The Search Efficiency of Theorem Proving Strategies.
In Bundy [1994].

Cesare Tinelli.
A DPLL-based calculus for ground satisfiability modulo theories.
In Giovambattista Ianni and Sergio Flesca, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (Cosenza, Italy)*, volume 2424 of *Lecture Notes in Artificial Intelligence*. Springer, 2002.

Adnan Yahya and David Plaisted.
Ordered Semantic Hyper-Tableaux.
Journal of Automated Reasoning, 29(1):17–57, 2002.