Instance Based Methods

TABLEAUX 2005 Tutorial (Koblenz, September 2005)

Peter Baumgartner

Max-Planck-Institut für Informatik

Saarbrücken, Germany

`http://www.mpi-sb.mpg.de/~baumgart/`

Gernot Stenz

Technische Universität München, Germany

`http://www4.in.tum.de/~stenzg`

Funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) under Verisoft project grant 01 IS C38

Purpose of Tutorial

Instance Based Methods (IMs): a family of calculi and proof procedures for first-order clause logic, developed during past ten years

Tutorial provides overview about the following

- **Common principles behind IMs, some calculi, proof procedures**
- **Comparison among IMs, difference from tableaux and resolution**
- **Ranges of applicability/non-applicability**
- **Improvements and extensions: universal variables, equality, . . .**
- **Picking up SAT techniques**
- **Implementations and implementation techniques**

Setting the Stage

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) it thus suffices to

- enumerate growing finite sets of such 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

Setting the Stage

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) it thus suffices to

- enumerate growing finite sets of such 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
$\forall x\, \exists y\, P(y,x)$ $\wedge \forall z\, \neg P(z,a)$	$\longrightarrow$	$P(f(x),x)$ $\neg P(z,a)$

Outer loop:
Grounding

Inner loop:
Propositional
DPLL

An early IM: the DPLL Procedure

Given Formula → **Clause Form**

Preprocessing:

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

$\longrightarrow$

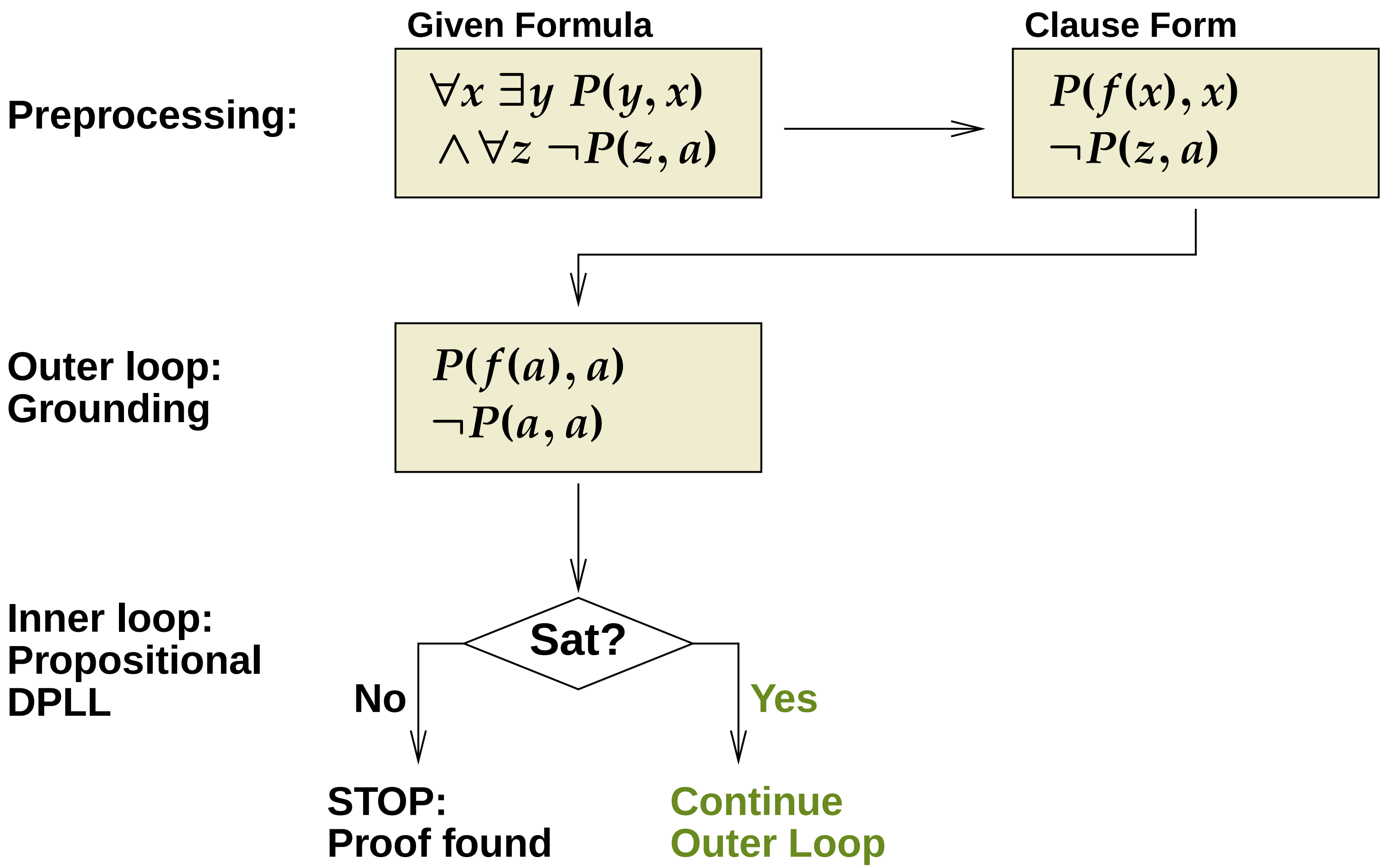
$P(f(x), x)$
$\neg P(z, a)$

Outer loop: Grounding

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

Inner loop: Propositional DPLL

An early IM: the DPLL Procedure



An early IM: the DPLL Procedure

	Given Formula		Clause Form
Preprocessing:	$\forall x \, \exists y \, P(y, x)$ $\land \forall z \, \neg P(z, a)$	$\longrightarrow$	$P(f(x), x)$ $\neg P(z, a)$
Outer loop: Grounding	$P(f(a), a)$ $\neg P(a, a)$	. . . .	$P(f(a), a)$ $\neg P(a, a)$ $\neg P(f(a), a)$

Inner loop: Propositional DPLL

An early IM: the DPLL Procedure

Preprocessing:

Given Formula

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

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

Sat?

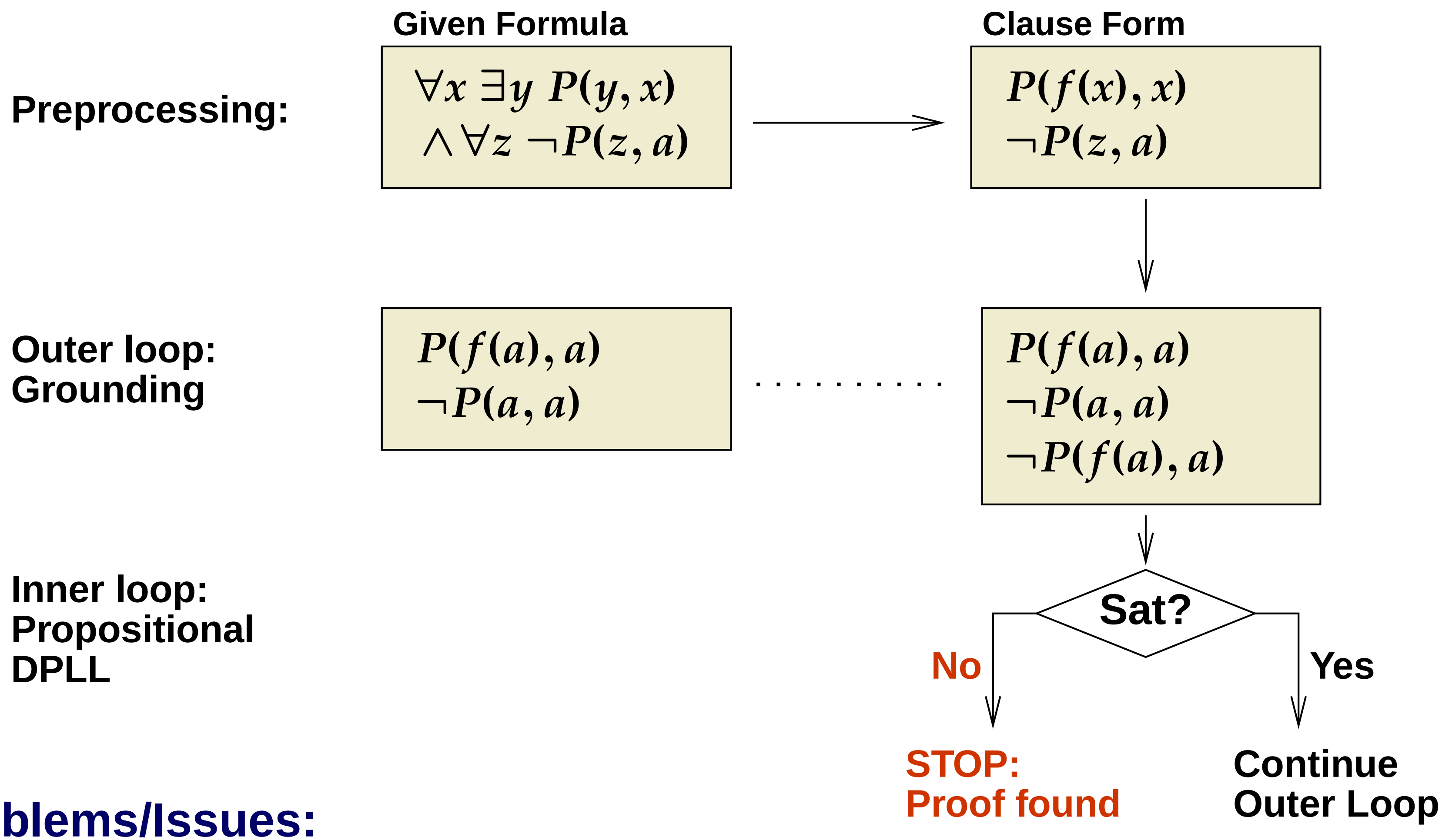
No

STOP:
Proof found

Yes

Continue
Outer Loop

An early IM: the DPLL Procedure



Problems/Issues:

- Controlling the 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
- We try to work out common principles and also differences
- Comparison with Resolution and Tableaux
- Applicability/Non-Applicability

Development of IMs (I)

Purpose of this slide

- 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 (II)

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]

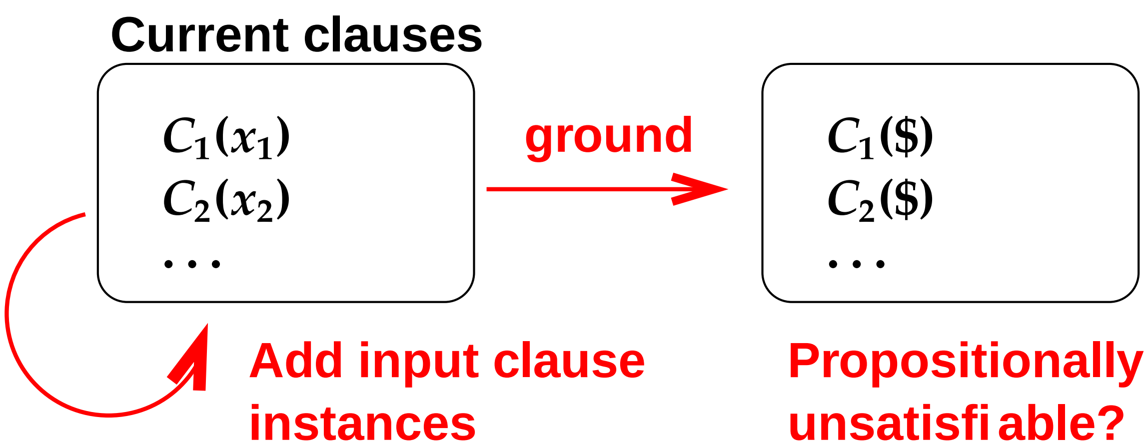
Ginsberg & Parkes method [Ginsberg and Parkes, 2000]

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. $\forall x P(x)$?

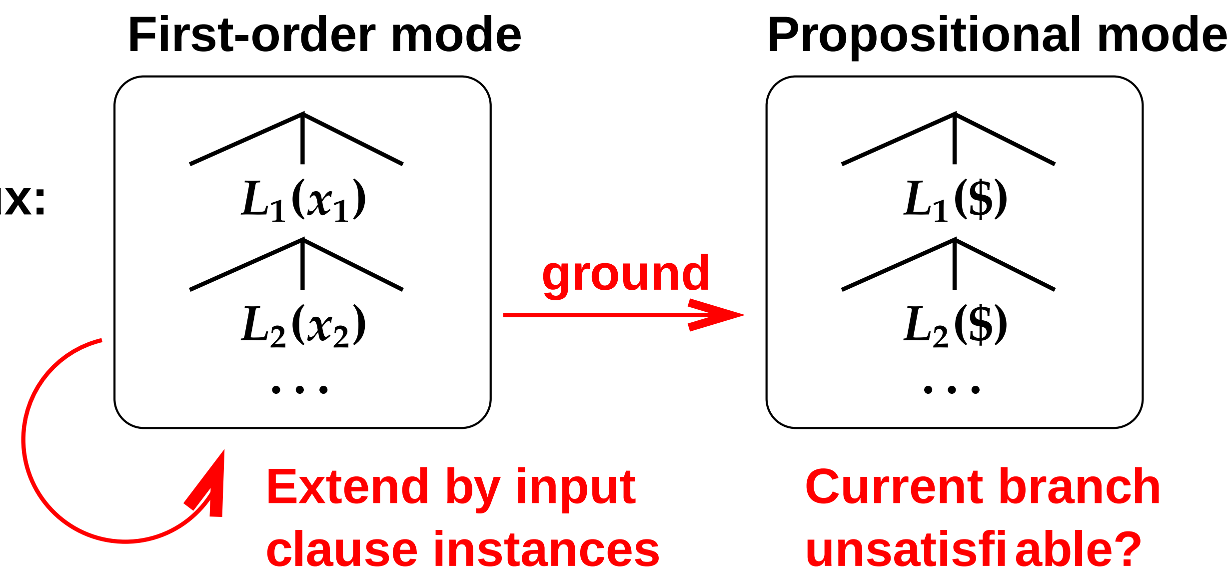


Two-Level vs. One-Level Calculi

One-Level Calculi

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

E.g. Tableaux:

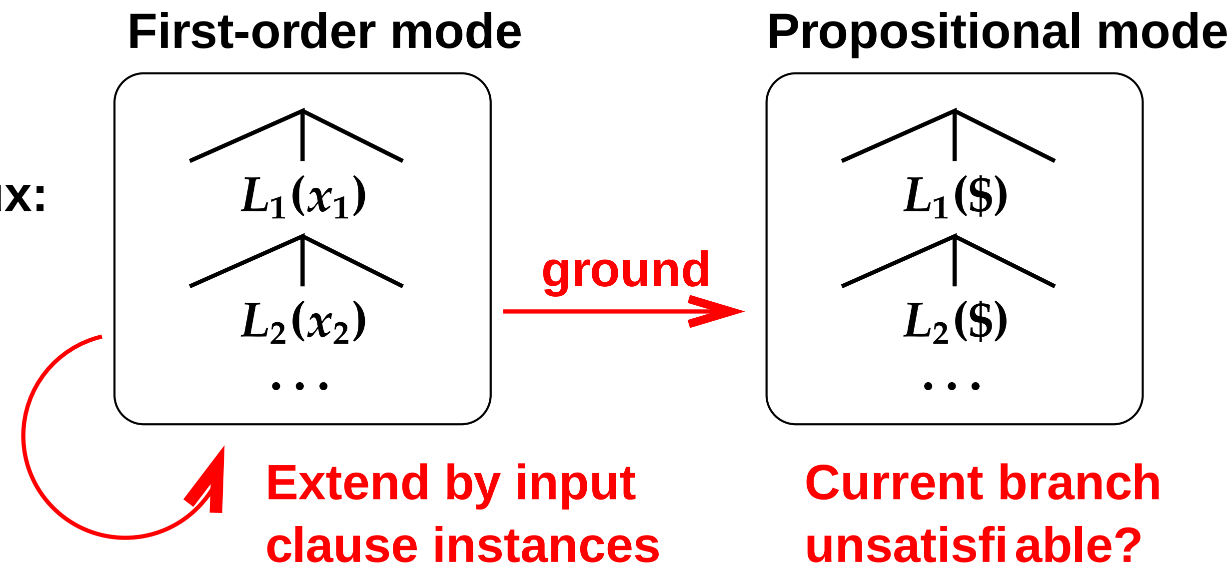


Two-Level vs. One-Level Calculi

One-Level Calculi

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

E.g. Tableaux:



Next: two-level calculus "Inst-Gen"

Inst-Gen

- We have chosen Inst-Gen for presentation because of its elegance and simplicity
- Talk proceeds with
  - Idea behind Inst-Gen

    (it provides a clue to the working of two-level calculi)
  - Inst-Gen calculus
  - Comparison to Resolution
  - Mentioning some improvements, as justified by “idea behind”
- See [Ganzinger and Korovin, 2003] for details

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: $\perp$ **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**

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

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 <u>selected</u> 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:

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

$$\underline{\neg P(a)} \qquad\qquad \underline{\neg P(a)}$$

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 <u>selected</u> in S by $I_\perp$ is not a model of S

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

Inst-Gen - Underlying Idea (III)

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

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

$$P(a) \vee \underline{Q(a)} \qquad\qquad P(a) \vee \underline{Q(a)}$$

$$\underline{\neg P(a)} \qquad\qquad \underline{\neg P(a)}$$

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 $P(a) \vee Q(a)$

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

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

How to derive candidate model I_S?

Inst-Gen - Model Construction

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

$$S: \quad \underline{P(x)} \vee Q(x) \qquad\qquad \Sigma = \{a, b\},\ S_{\text{ground}}: \quad \underline{P(b)} \vee Q(b)$$

$$P(a) \vee \underline{Q(a)} \qquad\qquad P(a) \vee \underline{Q(a)}$$

$$\underline{\neg P(a)} \qquad\qquad \underline{\neg P(a)}$$

- **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

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

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

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

Example:

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

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

(ii) θ is a proper instantiator

Inst-Gen - Outer Loop

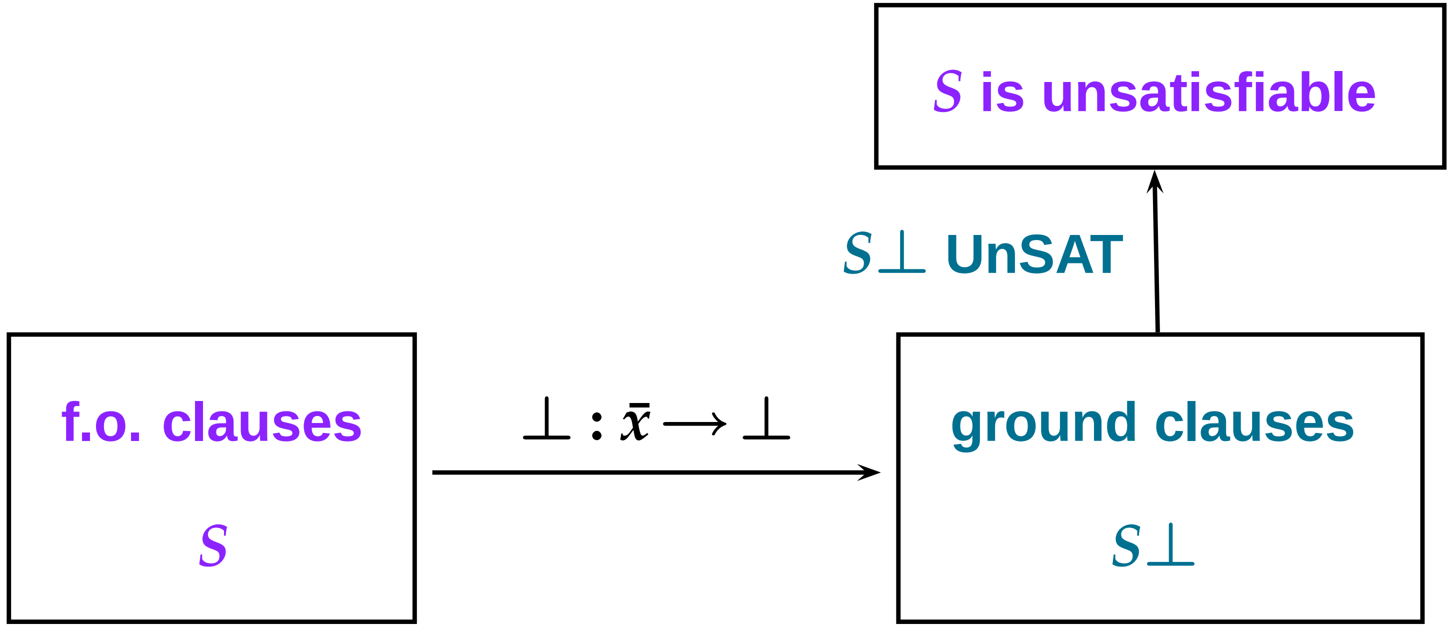
f.o. clauses

S

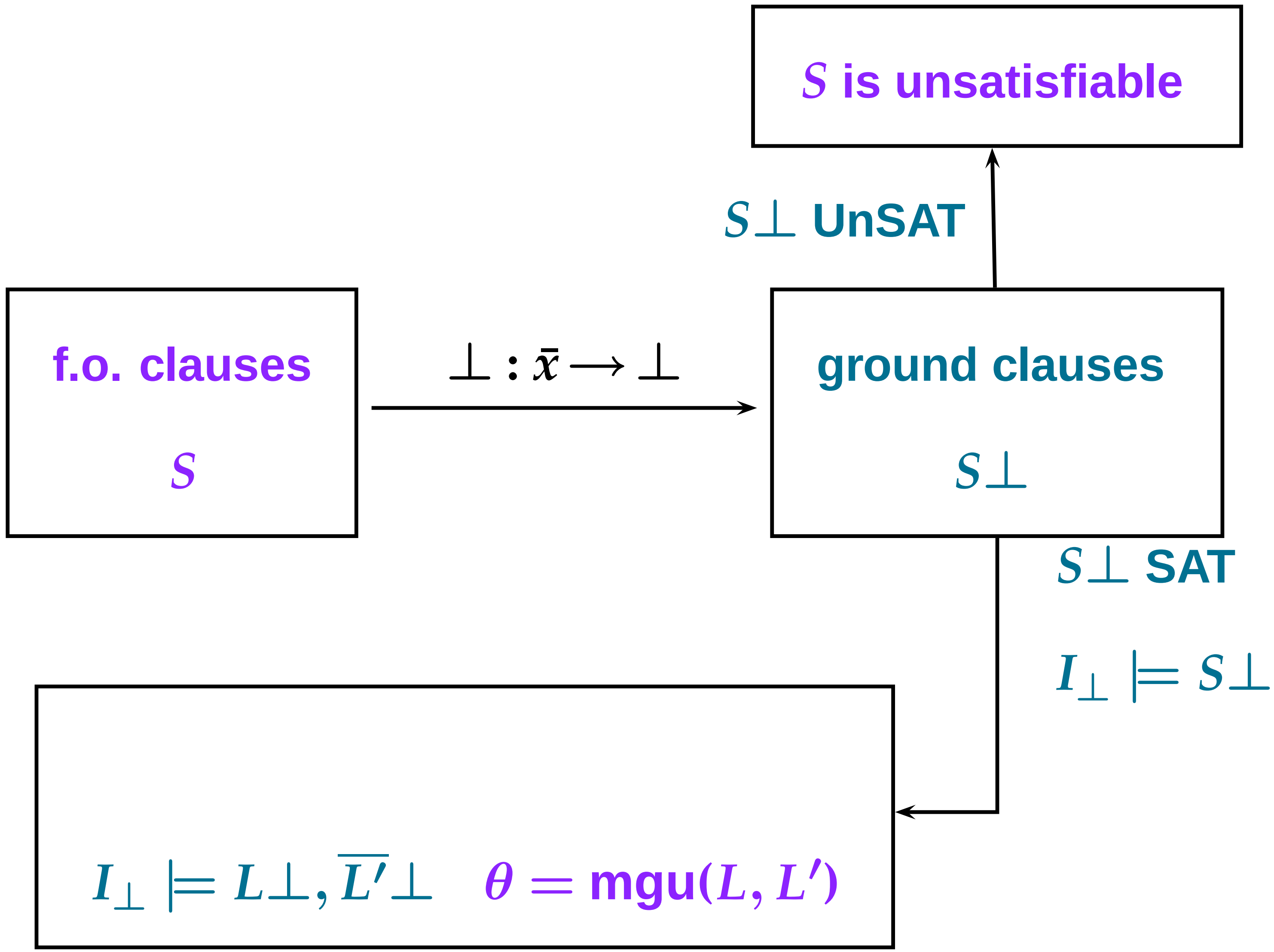
Inst-Gen - Outer Loop

$$\boxed{\begin{array}{c}\text{f.o. clauses}\\ S\end{array}} \xrightarrow{\perp : \bar{x} \to \perp} \boxed{\begin{array}{c}\text{ground clauses}\\ S\perp\end{array}}$$

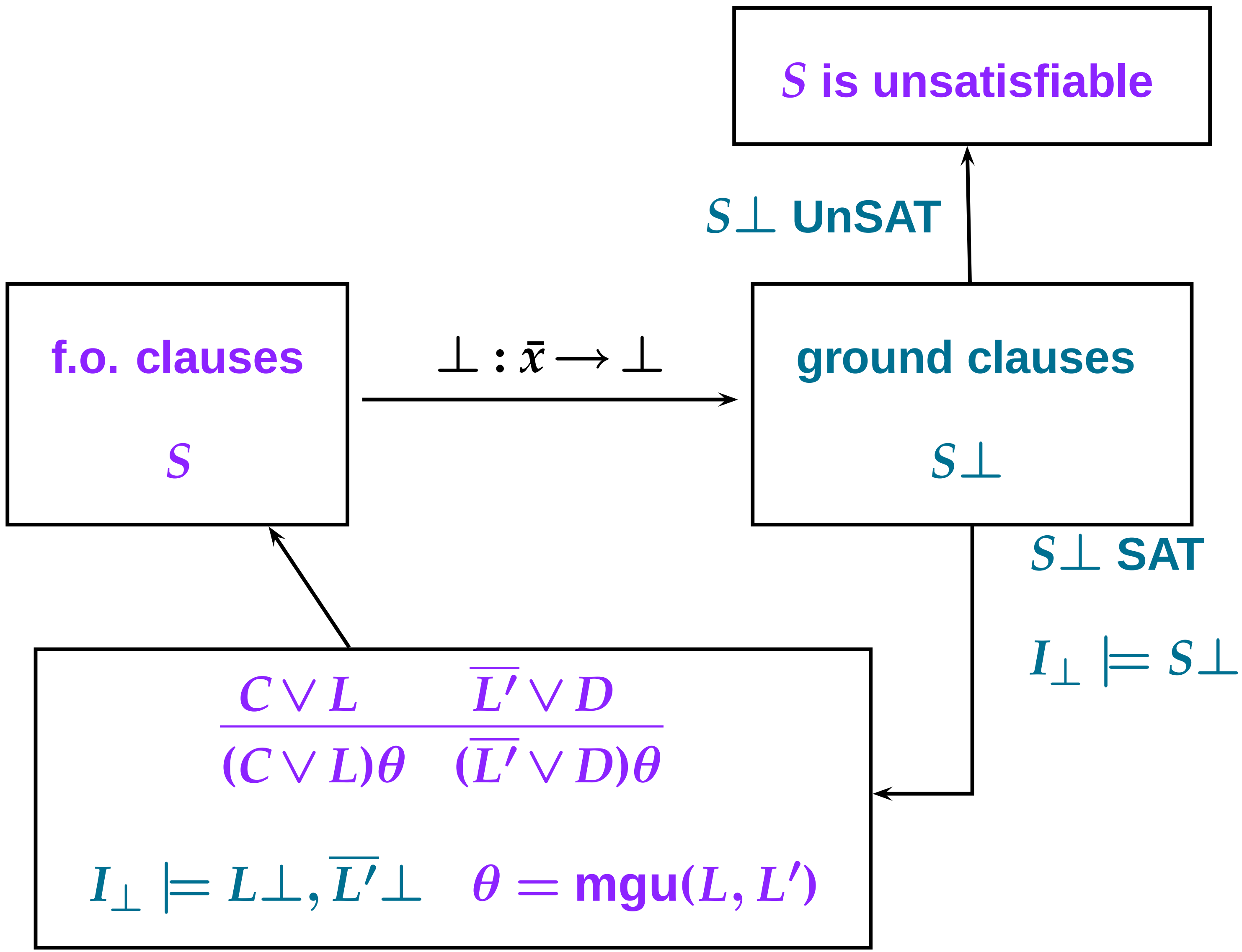
Inst-Gen - Outer Loop



Inst-Gen - Outer Loop



Inst-Gen - Outer Loop



Properties and Improvements

- As efficient as possible in propositional case
- <u>Literal selection</u> *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

$$\frac{(C \vee L) \qquad (\overline{L'} \vee D)}{(C \vee D)\theta}$$

$$\theta = \mathrm{mgu}(L, L')$$

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

Inst-Gen

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

$$\theta = \mathrm{mgu}(L, L')$$

- 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 provers use Resoluti

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 model with finite domain
- 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]
- See also important paper [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
- Stepwisely 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 := $\{\}$;; Set of current ground instances of clauses of 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 $I \models G$
od
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 $(\overline{L_1}, \ldots, \overline{L_m})$
- Guess δ s.th. $I_0(L_1, \ldots, L_m) \not\models C\gamma$, where $\gamma = \theta\delta$

Search and Update in OSHL

$I_o = \{Ra\}$

(all other atoms false)

S:

(1) $R(a) \leftarrow$

(2) $P(x) \leftarrow R(a)$

(3) $R(y) \vee Q(x,y) \leftarrow P(x)$

(4) $\leftarrow Q(a,c)$

(5) $\leftarrow R(c)$

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

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

Why?

- Migrate to the first-order level those very effective techniques developed for propositional DPLL
- 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)
- *Theorem Proving:* alternative to established methods
- *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) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$

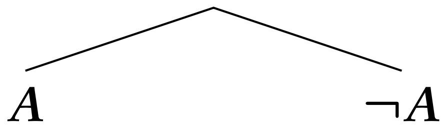
$\langle$**empty tree**$\rangle$

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

- **A Branch stands for an interpretation**
- ***Purpose of splitting:*** **satisfy a clause that is currently 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)** $\neg D \lor \neg B$

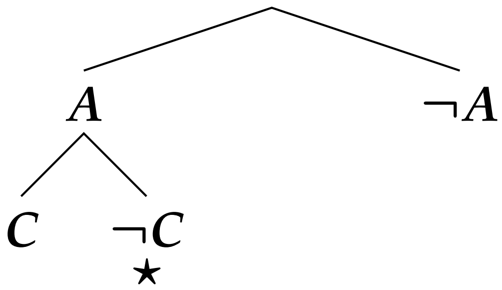


$$\{A\} \models A \lor B$$
$$\{A\} \not\models C \lor \neg A$$
$$\{A\} \models D \lor \neg C \lor \neg A$$
$$\{A\} \models \neg D \lor \neg B$$

- **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) $\neg D \vee \neg B$

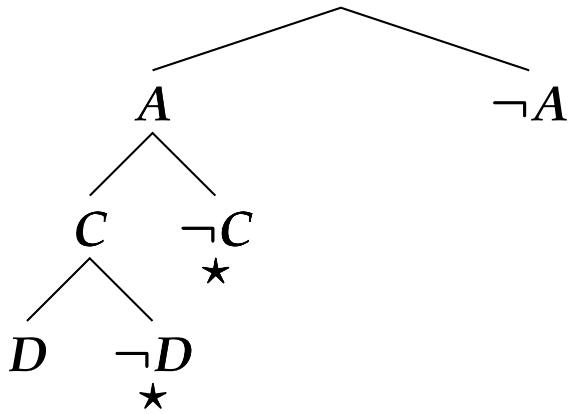


$$\{A, C\} \models A \vee B$$
$$\{A, C\} \models C \vee \neg A$$
$$\{A, C\} \not\models D \vee \neg C \vee \neg A$$
$$\{A, C\} \models \neg D \vee \neg B$$

- 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)** $\neg D \vee \neg B$



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

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

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

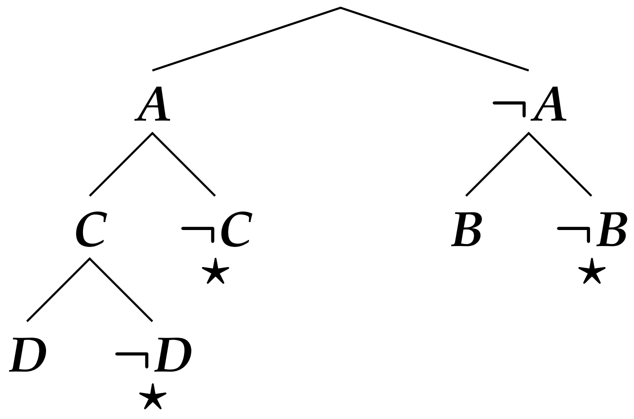
$\{A, C, D\} \models \neg D \vee \neg B$

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.

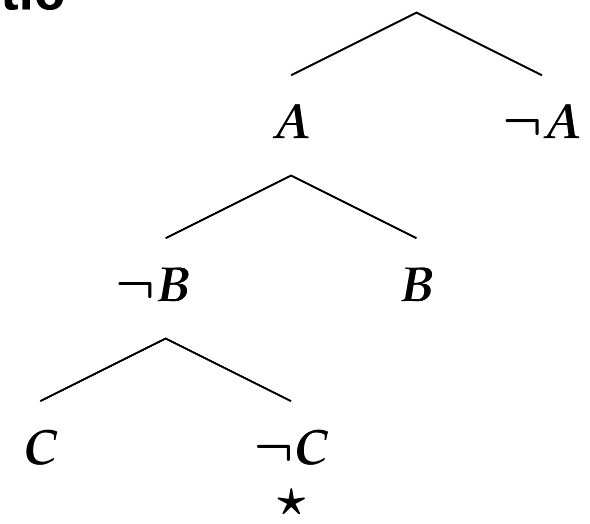
- **A Branch stands for an interpretation**
- ***Purpose of splitting:* satisfy a clause that is currently falsified**
- **Close branch if some clause is plainly falsified by it ($\star$)**

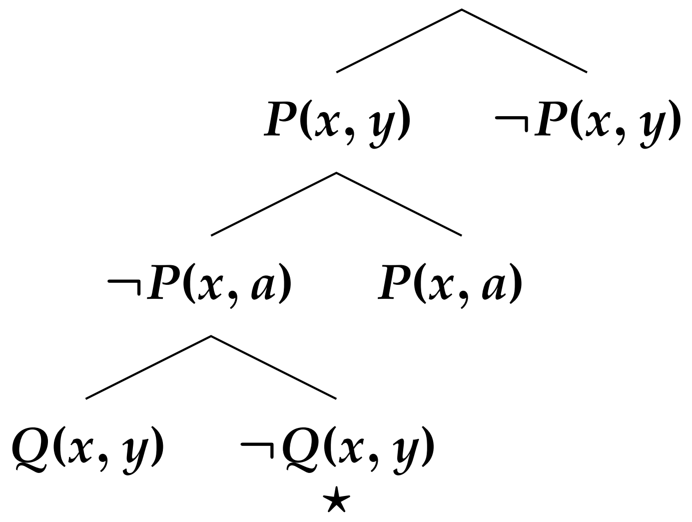
Meta-Level Strategy

Lifted data structures:

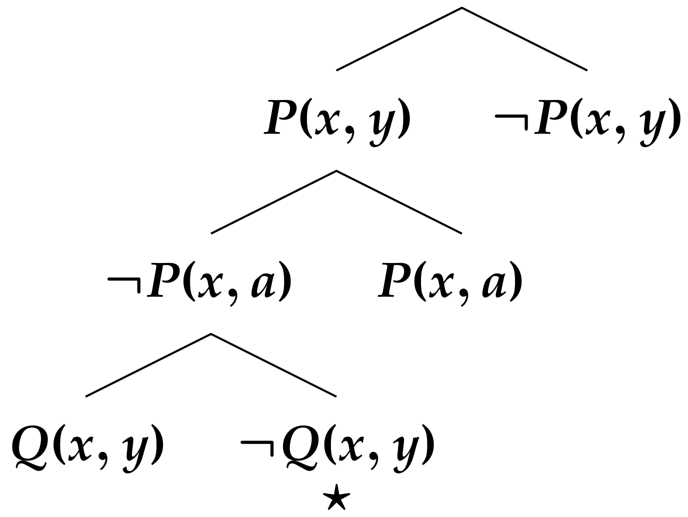
	DPLL	FDPLL
Clauses	$B \vee C$	$P(x, y) \vee Q(x, x)$

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 $I_B = \{\ldots\}$:

- **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 $I_B = \{\ldots\}$:

$P(x, y)$ $\longrightarrow$

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

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

- **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

Branch B:

$P(x, y)$

$\neg P(a, y)$

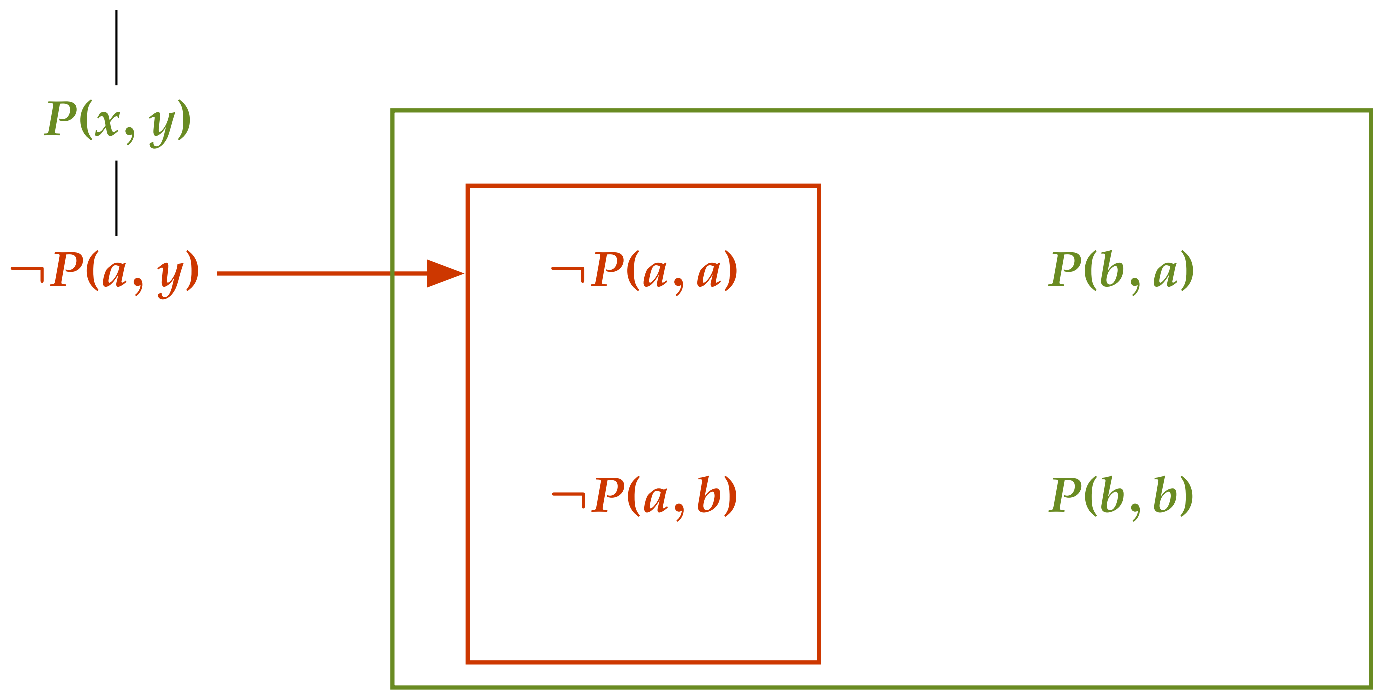
Interpretation $I_B = \{\ldots\}$:

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

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

Extracting an Interpretation from a Branch

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



- **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:

$P(x, y)$

$\neg P(a, y)$

$\neg P(b, b)$

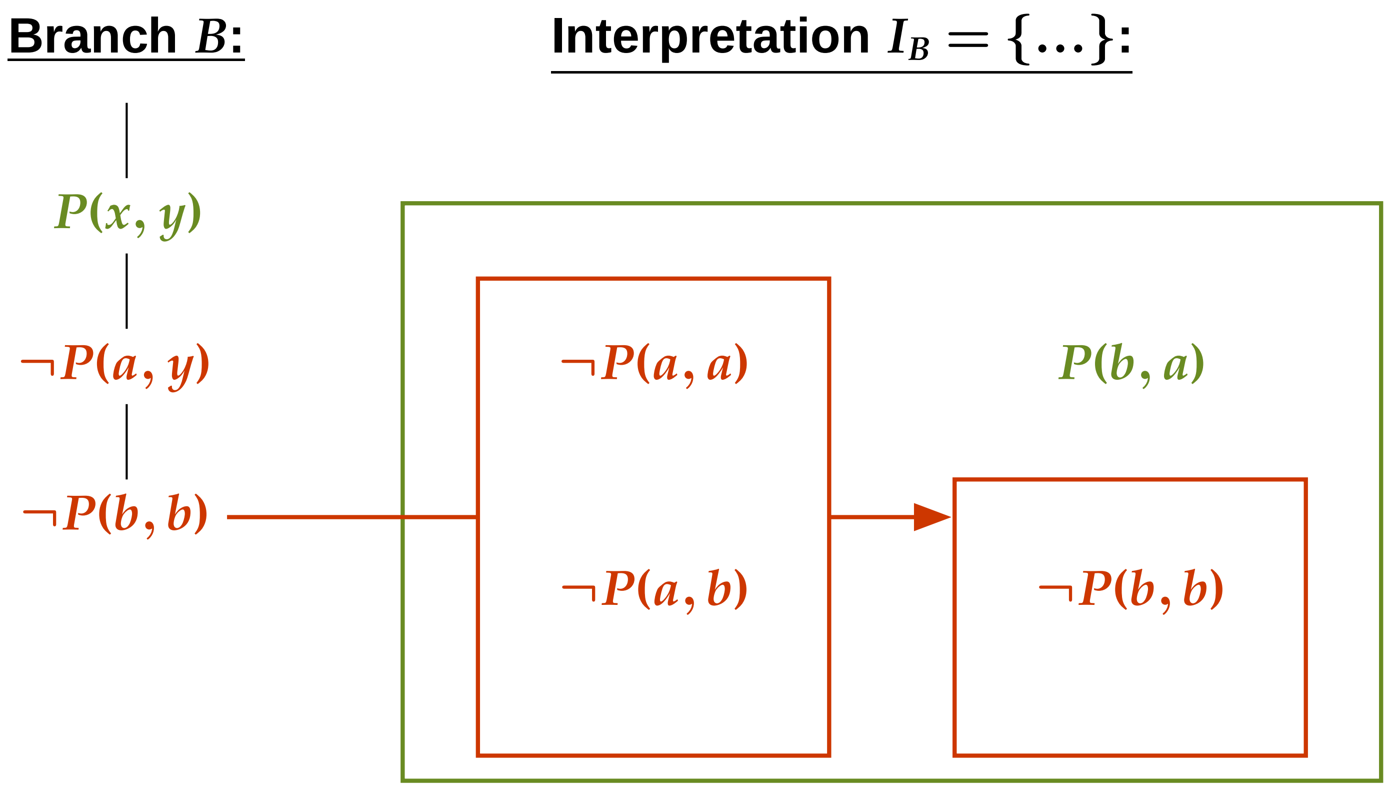
Interpretation $I_B = \{\ldots\}$:

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

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

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

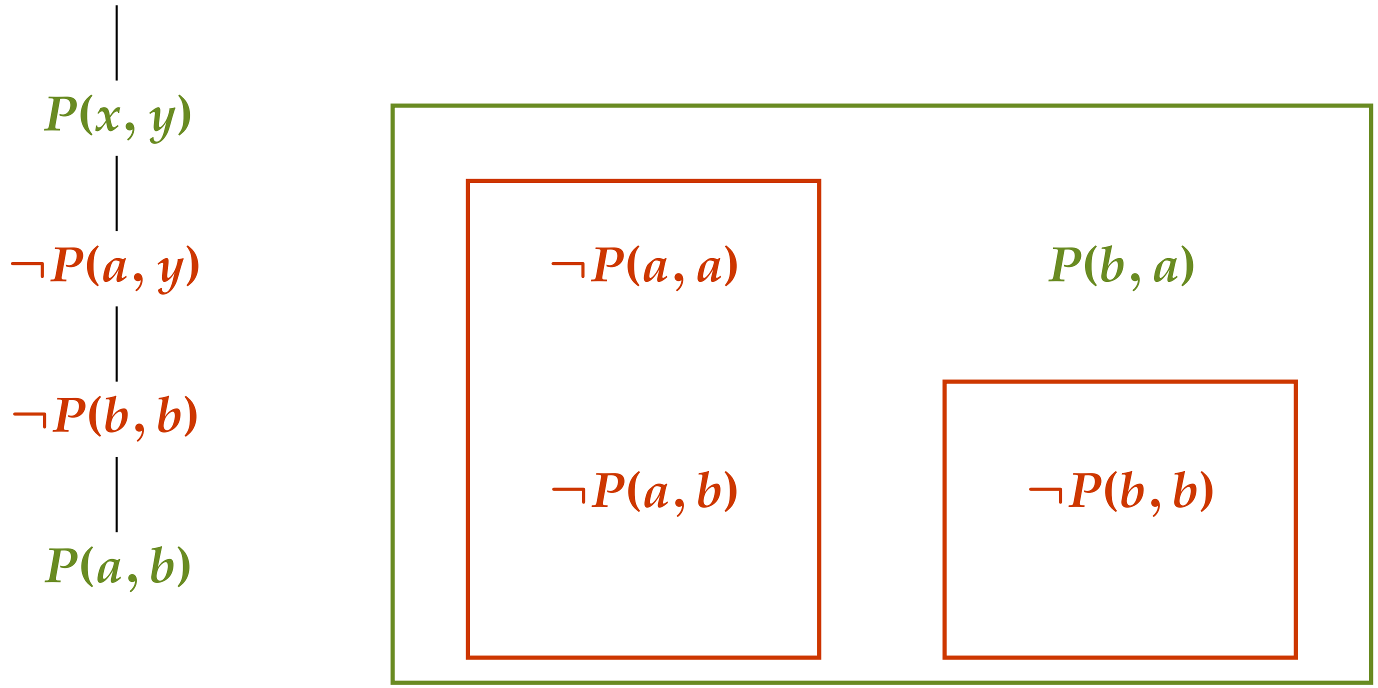
Extracting an Interpretation from a Branch



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

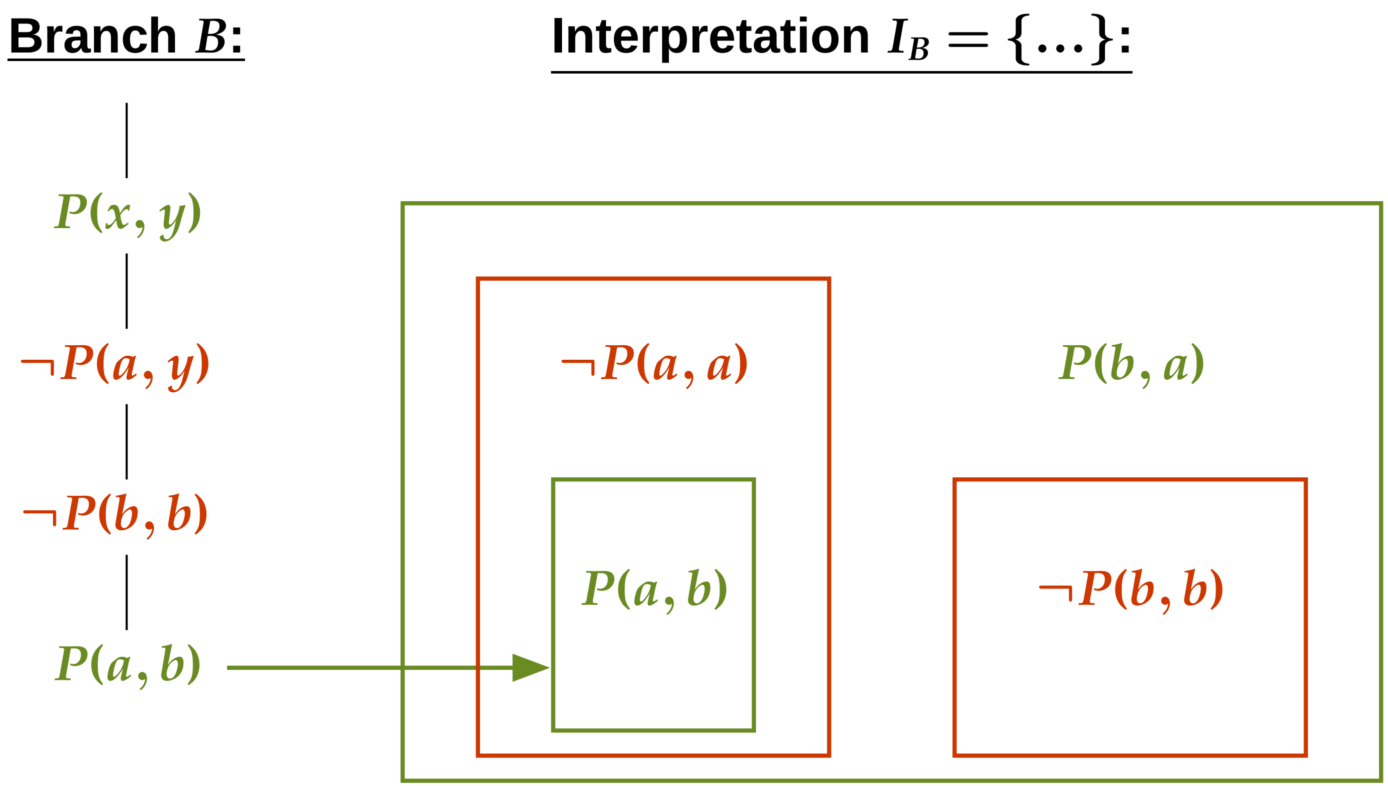
Extracting an Interpretation from a Branch

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



- **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:

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

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

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

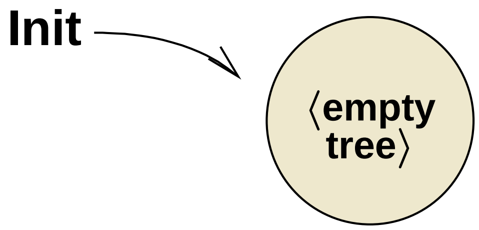
- A branch literal specifies the truth values for all its ground instances, unless there is a more specific literal specifying the opposite truth value
- The order of literals does not matter

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:

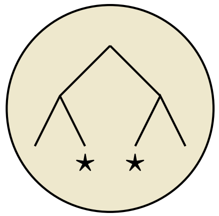


FDPLL Calculus - Main Loop

Input: **a clause set S**

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

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

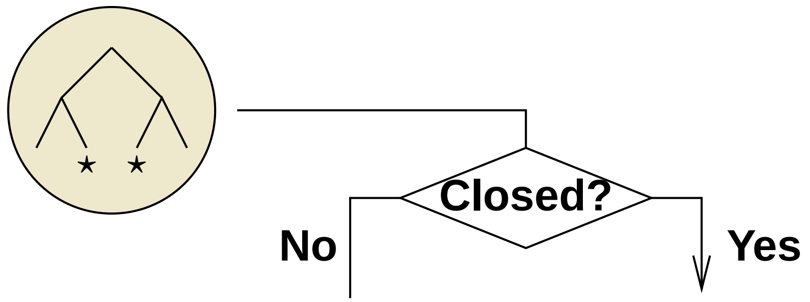


FDPLL Calculus - Main Loop

Input: **a clause set** S

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

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

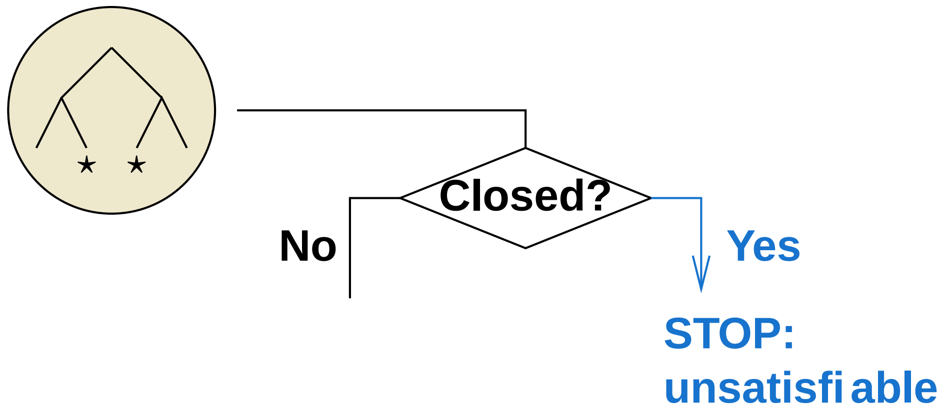


FDPLL Calculus - Main Loop

Input: **a clause set** S

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

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

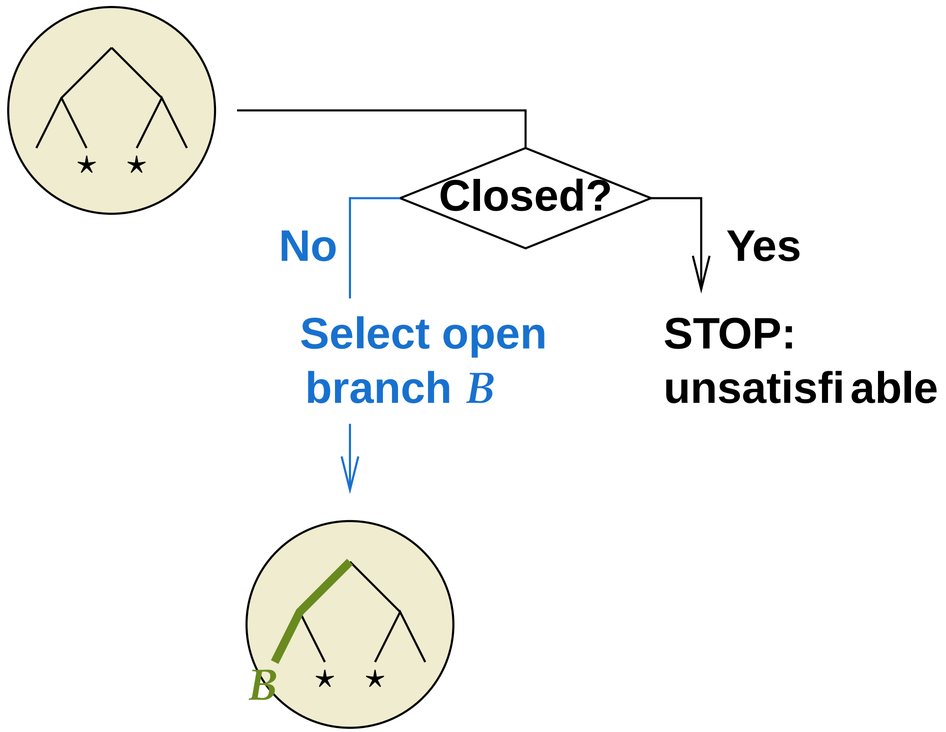


FDPLL Calculus - Main Loop

Input: **a clause set S**

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

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

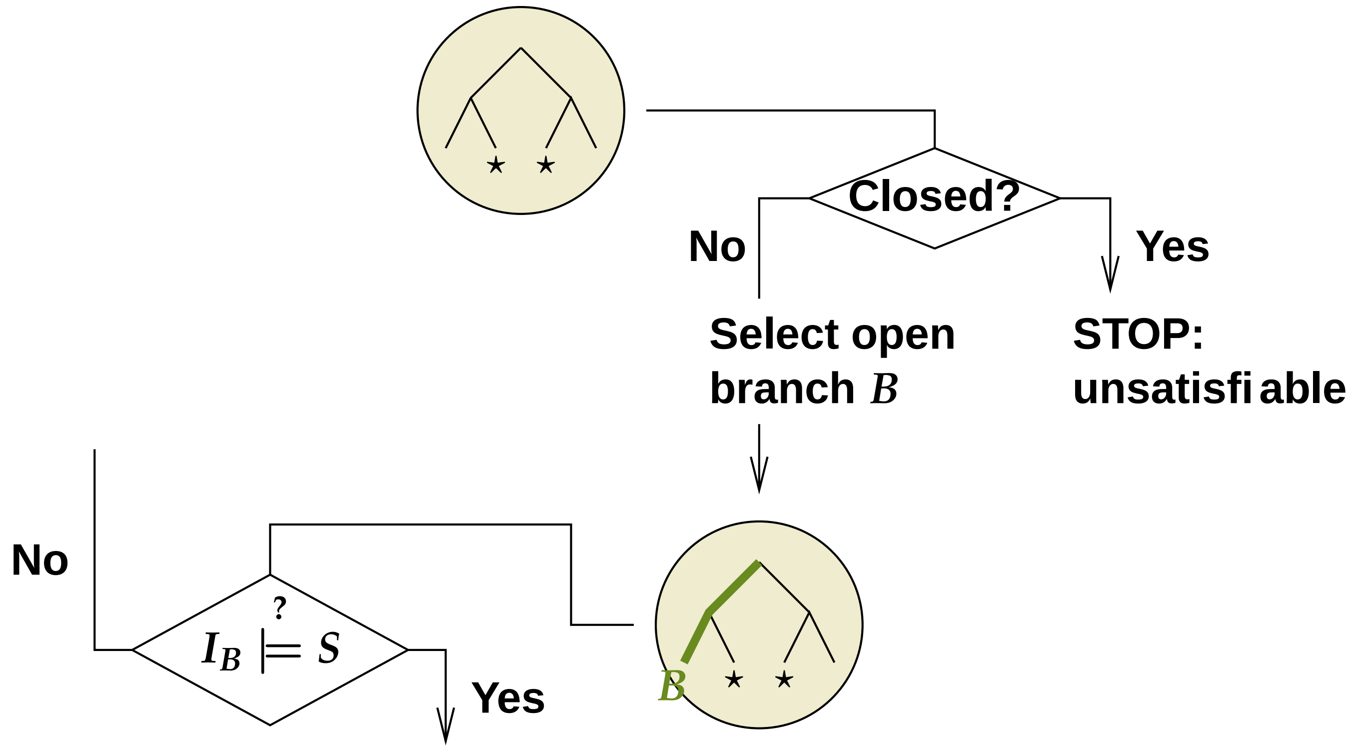


FDPLL Calculus - Main Loop

Input: **a clause set S**

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

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

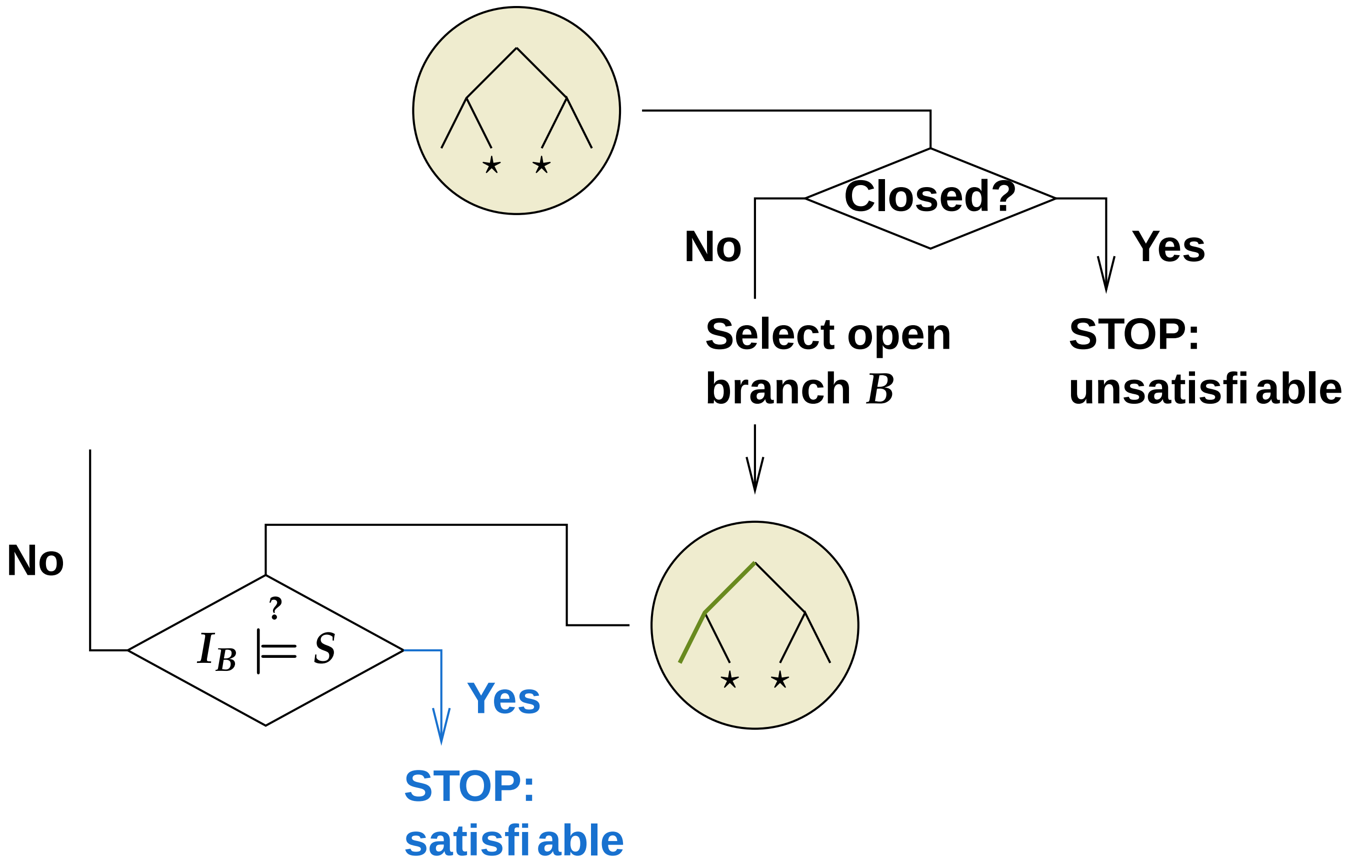


FDPLL Calculus - Main Loop

Input: a clause set S

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

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

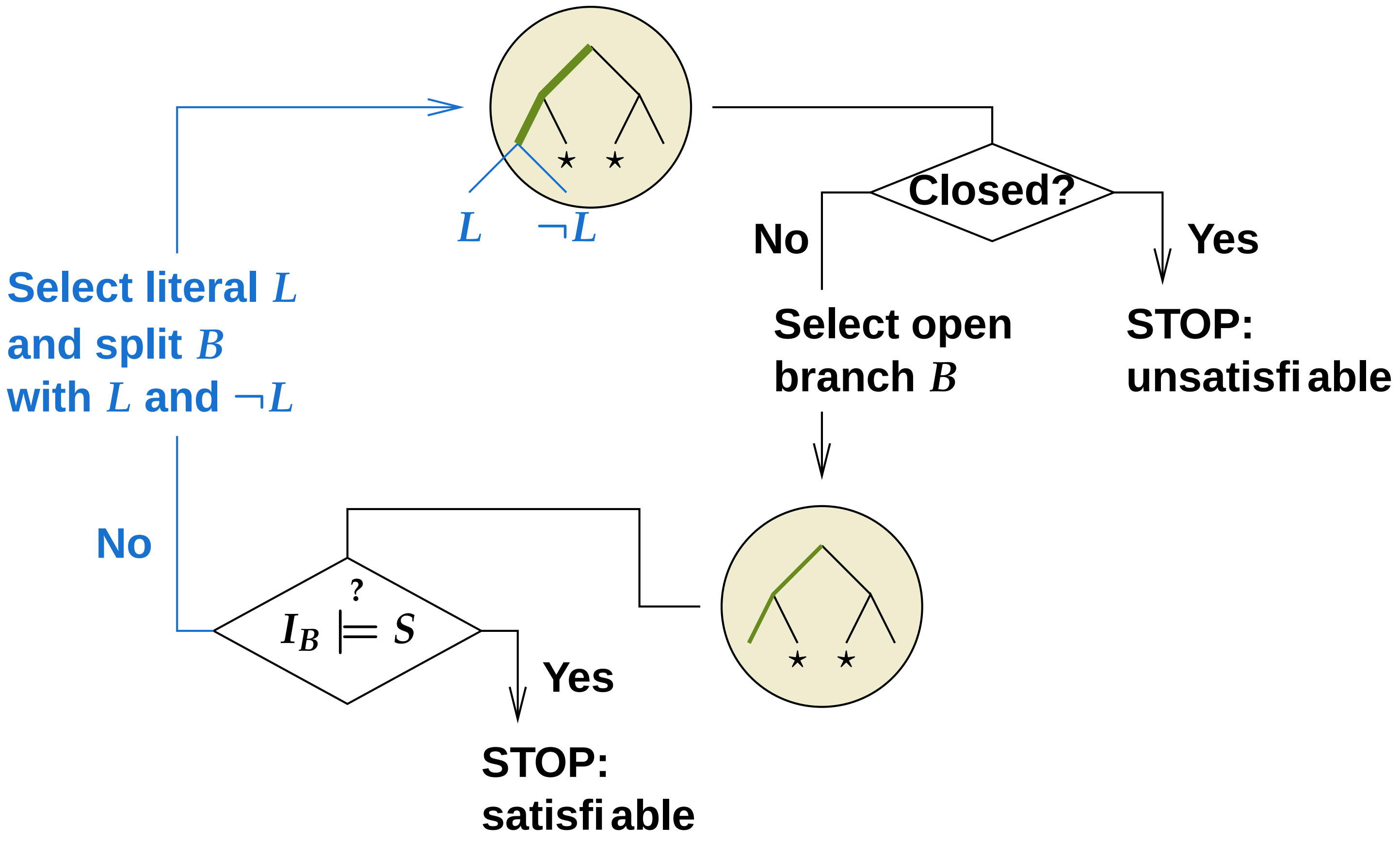


FDPLL Calculus - Main Loop

Input: a clause set S

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

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

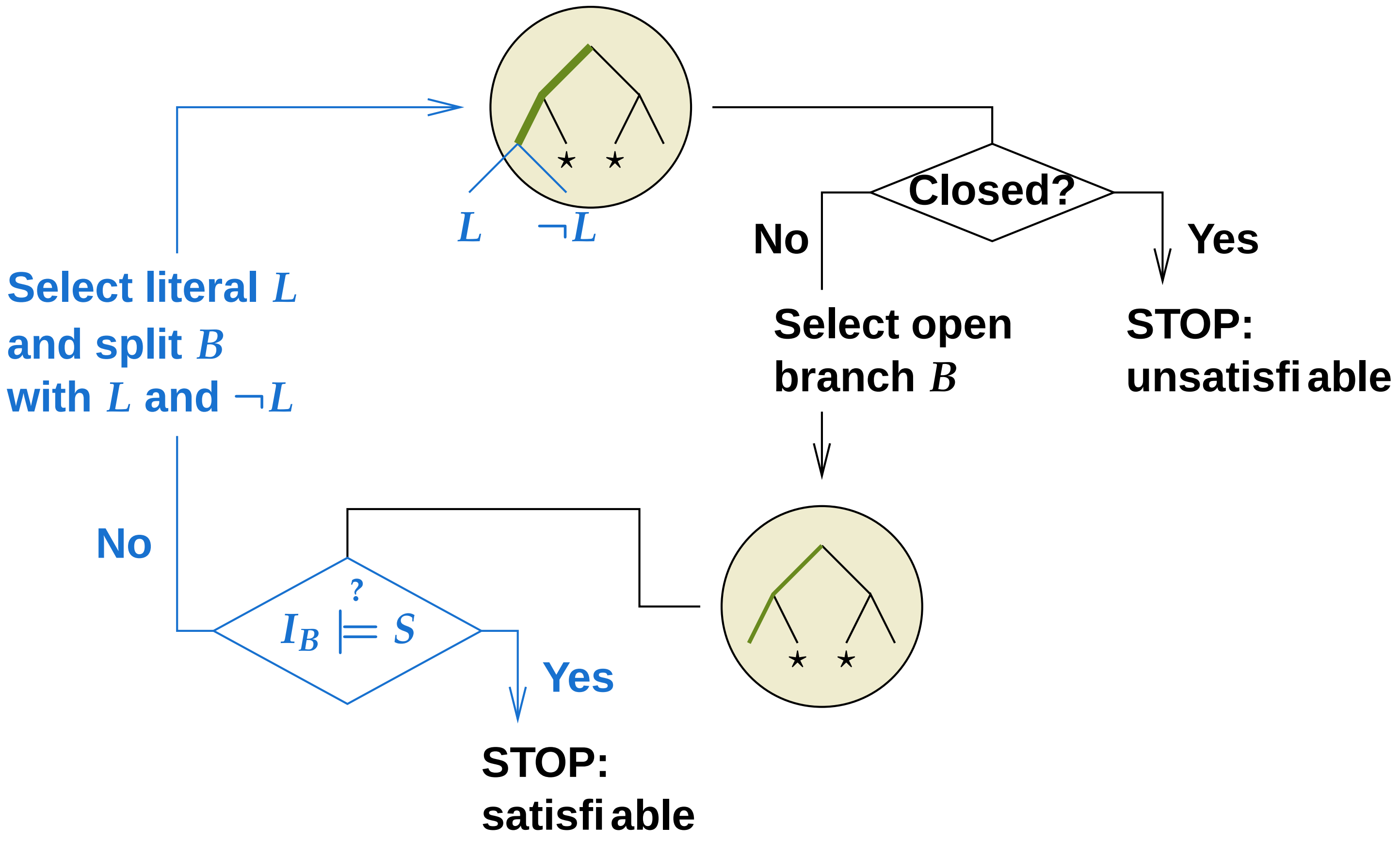


FDPLL Calculus - Main Loop

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

```
(1)  train(X,Y) ; flight(X,Y).       %% train from X to Y or flight from X to Y.
(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 relation.
                     connect(Y,Z).
```

FDPLL – Model Computation Example

```
(1)  train(X,Y) ; flight(X,Y).     %% train from X to Y or flight from X to Y.
(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 relation.
                     connect(Y,Z).
```

Computed Model (as output by Darwin implementation)

```
+ 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: $\boldsymbol{train(x,y) \lor flight(x,y)}$

FDPLL Model Computation Example - Derivation

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

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

FDPLL Model Computation Example - Derivation

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

$$\neg \mathit{flight}(sb, x) \qquad \mathit{flight}(sb, x)$$

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

FDPLL Model Computation Example - Derivation

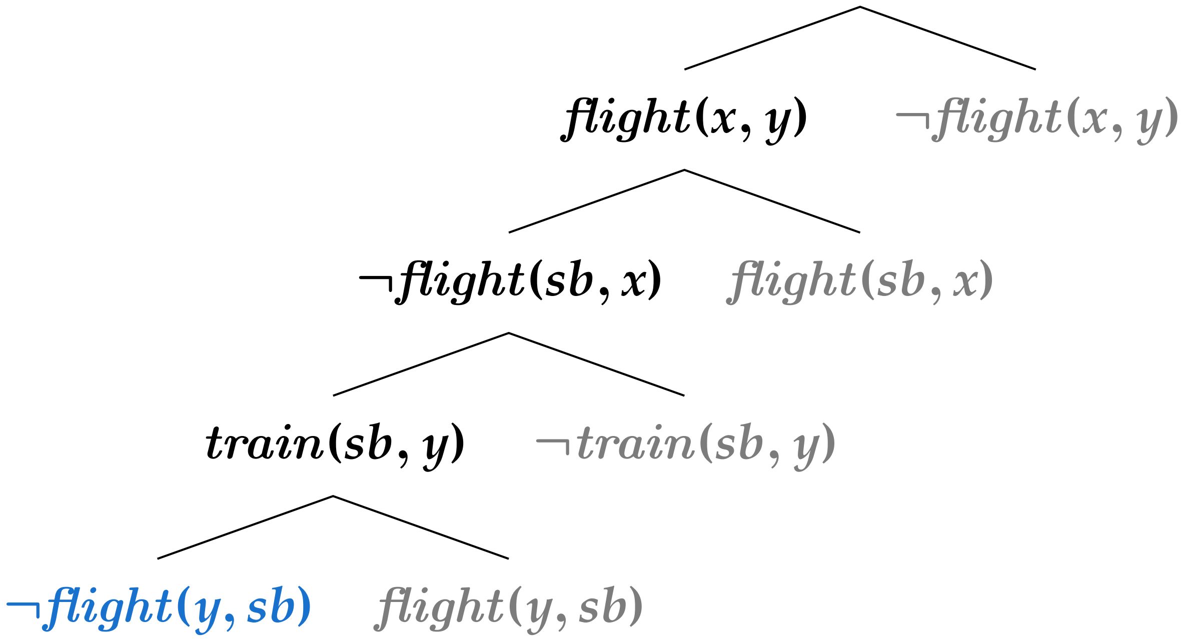
$$\mathit{flight}(x, y) \qquad \neg \mathit{flight}(x, y)$$

$$\neg \mathit{flight}(sb, x) \qquad \mathit{flight}(sb, x)$$

$$\mathit{train}(sb, y) \qquad \neg \mathit{train}(sb, y)$$

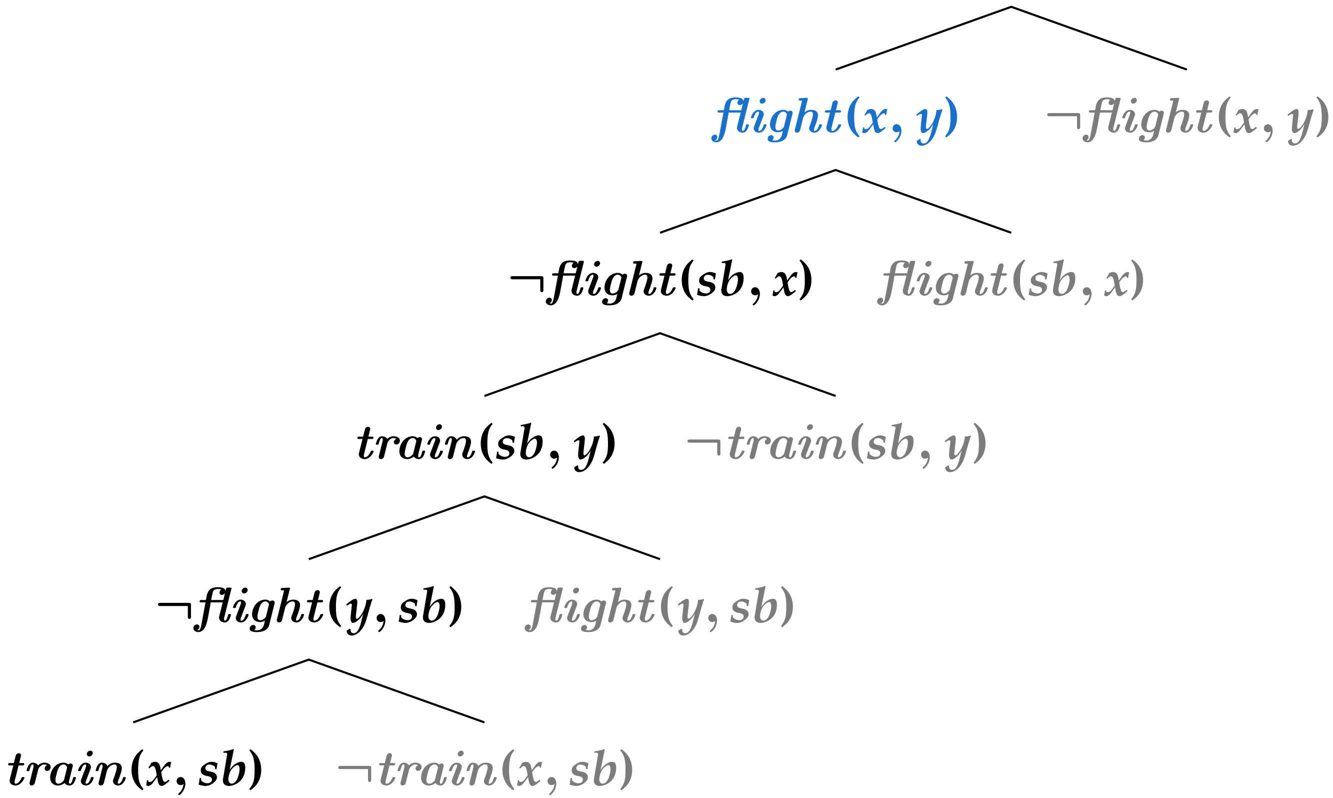
Clause instance used in inference: $\mathit{flight}(sb, y) \vee \neg \mathit{flight}(y, sb)$

FDPLL Model Computation Example - Derivation



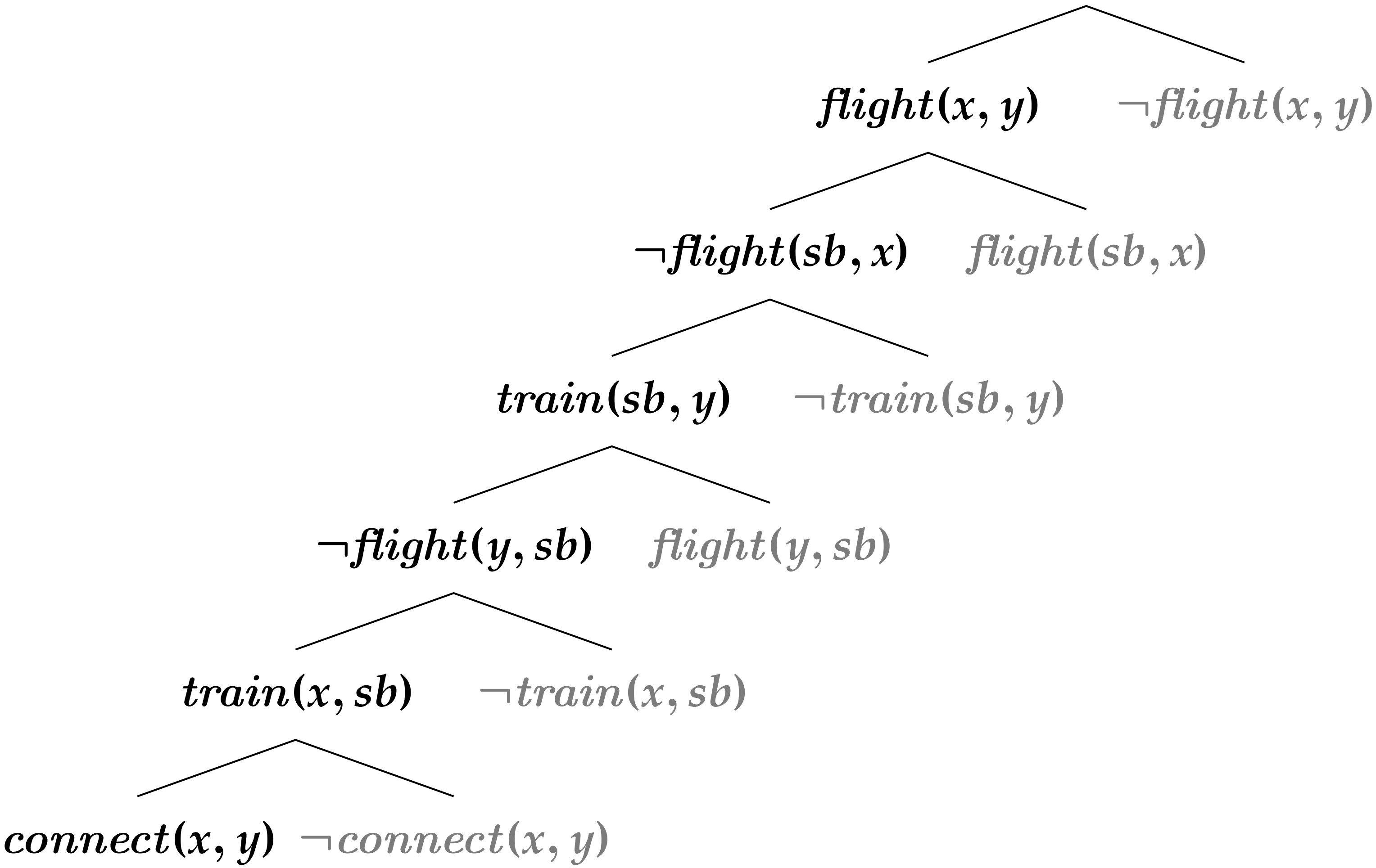
Clause instance used in inference: $train(x, sb) \vee flight(x, sb)$

FDPLL Model Computation Example - Derivation



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

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 wouldn't call it "extension"**
- **Extension of Tinelli's sequent-style DPLL [Tinelli, 2002]**
- **See [Baumgartner and Tinelli, 2003] for calculus, [Baumgartner *et al.*, 2005] 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**

Model Evolution Calculus – Data Structure

- **Branches and clause sets may shrink as the derivation proceeds**
- **Such dynamics is best modeled with a sequent style calculus:**

$$\Lambda \vdash \Phi$$

Current Clause Set

Context: **A set of literals**
(the „current branch“)

Derivation Rules

- **Simplification rules**
- **Split**
- **Close**

Model Evolution Calculus – Data Structure

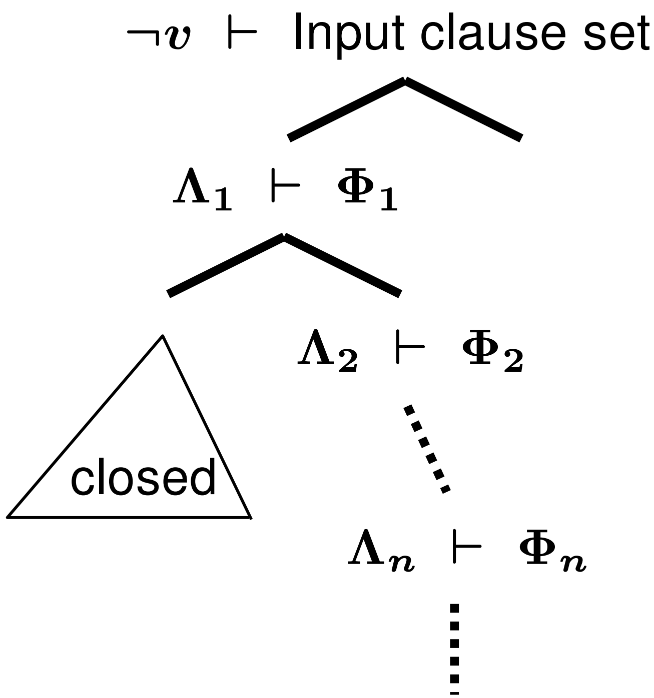
- **Branches and clause sets may shrink as the derivation proceeds**
- **Such dynamics is best modeled with a sequent style calculus:**

$$\Lambda \;\vdash\; \Phi$$

Current Clause Set

Context: **A set of literals (the „current branch“)**

Derivations



Derivation Rules

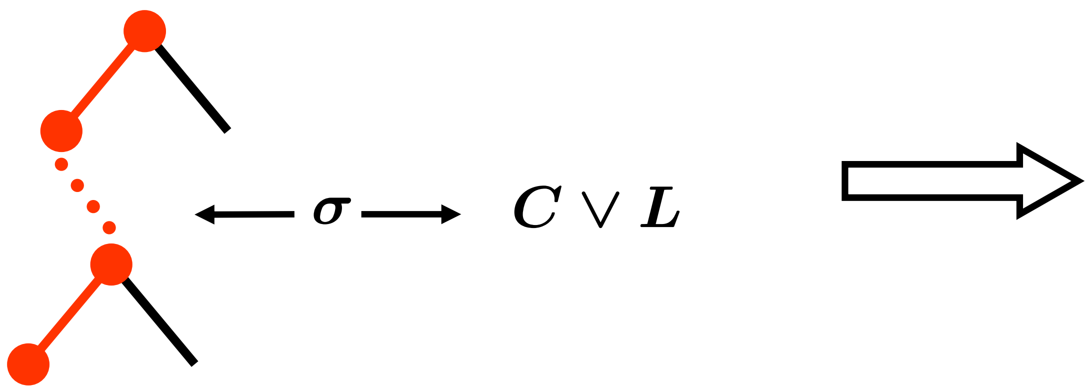
- **Simplification rules**
- **Split**
- **Close**

Derivation Rules - Split

$$\textbf{Split}\quad \frac{\Lambda \;\vdash\; \Phi,\, C \lor L}{\Lambda,\, L\sigma \;\vdash\; \Phi,\, C \lor L \qquad \Lambda,\, \overline{L}\sigma \;\vdash\; \Phi,\, C \lor L}$$

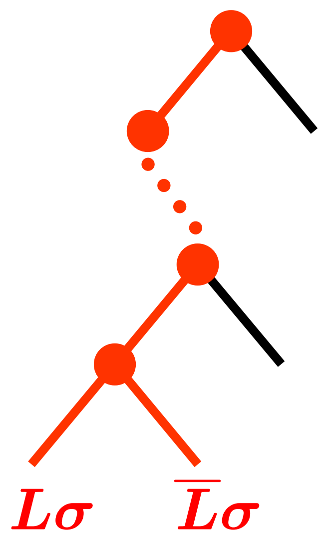
if

1. σ is a simultaneous mgu of $C \lor L$ against Λ,
2. neither $L\sigma$ nor $\overline{L}\sigma$ is contained in Λ, and
3. $L\sigma$ contains no variables (schematic variables OK, for simplicity here)



Λ current context

$\Phi, C \lor L$ current clause set



Derivation Rules – Split Example

$$\textbf{Split}\ \frac{\Lambda \ \vdash \ \Phi,\ C \vee L}{\Lambda,\ L\sigma \ \vdash \ \Phi,\ C \vee L \qquad \Lambda,\ \overline{L}\sigma \ \vdash \ \Phi,\ C \vee L}$$

if

1. σ is a simultaneous mgu of $C \vee L$ against Λ,
2. neither $L\sigma$ nor $\overline{L}\sigma$ is contained in Λ, and
3. $L\sigma$ contains no variables (schematic variables OK, for simplicity here)

$$\Lambda: \quad P(u, u) \qquad Q(v, b)$$

$$C \vee L : \neg P(x, y) \vee \neg Q(a, z)$$

Derivation Rules – Split Example

$$\textbf{Split}\ \frac{\Lambda \ \vdash\ \Phi,\ C \vee L}{\Lambda,\ L\sigma\ \vdash\ \Phi,\ C \vee L \qquad \Lambda,\ \overline{L}\sigma\ \vdash\ \Phi,\ C \vee L}$$

if

1. σ is a simultaneous mgu of $C \vee L$ against Λ,
2. neither $L\sigma$ nor $\overline{L}\sigma$ is contained in Λ, and
3. $L\sigma$ contains no variables (schematic variables OK, for simplicity here)

$$\Lambda: \quad P(u,u) \qquad Q(v,b)$$

$$C \vee L : \neg P(x,y) \vee \neg Q(a,z)$$

$$\sigma = \{\, x \mapsto u,\ y \mapsto u,\ v \mapsto a,\ z \mapsto b \,\}$$

$$(C \vee L)\sigma : \neg P(x,x) \vee \neg Q(a,b)$$

Derivation Rules – Split Example

$$\textbf{Split}\ \frac{\Lambda \ \vdash\ \Phi,\ C \vee L}{\Lambda,\ L\sigma\ \vdash\ \Phi,\ C \vee L \qquad \Lambda,\ \overline{L}\sigma\ \vdash\ \Phi,\ C \vee L}$$

if

1. σ is a simultaneous mgu of $C \vee L$ against Λ,
2. neither $L\sigma$ nor $\overline{L}\sigma$ is contained in Λ, and
3. $L\sigma$ contains no variables (schematic variables OK, for simplicity here)

$$\Lambda\text{:} \quad \underline{P(u,u)} \qquad \underline{Q(v,b)}$$

$$C \vee L : \neg P(x,y) \vee \neg Q(a,z)$$

$$\sigma = \{\, x \mapsto u,\ y \mapsto u,\ v \mapsto a,\ z \mapsto b \,\}$$

$$(C \vee L)\sigma : \neg \underline{P(x,x)} \vee \neg \underline{Q(a,b)}$$

2. violated **2. satisfied**

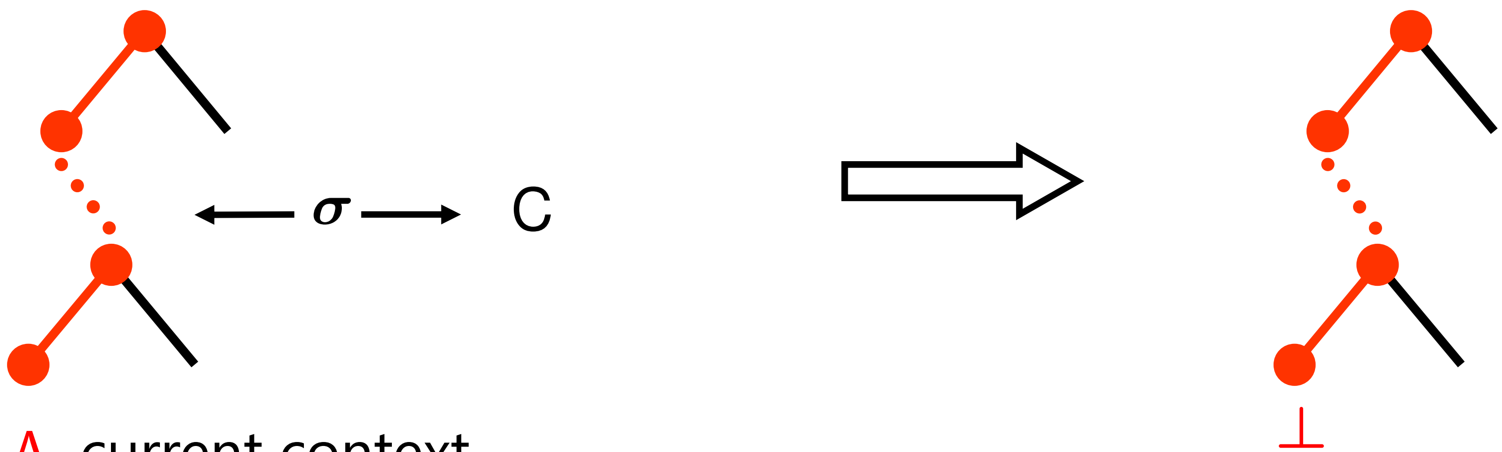
$L\sigma = \neg Q(a,b)$ is admissible for Split

Derivation Rules – Close

$$\textbf{Close}\ \frac{\Lambda \ \vdash \ \Phi,\ C}{\Lambda \ \vdash \ \bot}$$

if

1. $\Phi \neq \emptyset$ or $C \neq \bot$, and
2. there is a simultaneous mgu σ of C against Λ such that Λ contains the complement of each literal of $C\sigma$



Λ current context

Φ, C current clause set

Derivation Rules – Close Example

Close $\dfrac{\Lambda \;\vdash\; \Phi,\, C}{\Lambda \;\vdash\; \bot}$

if

1. $\Phi \neq \emptyset$ or $C \neq \bot$, and
2. there is a simultaneous mgu σ of C against Λ such that Λ contains the complement of each literal of $C\sigma$

Λ: $P(u, u) \quad Q(a, b)$

$C : \neg P(x, y) \vee \neg Q(a, z)$

Derivation Rules – Close Example

$$\textbf{Close}\ \frac{\Lambda \quad \vdash \quad \Phi,\ C}{\Lambda \quad \vdash \quad \bot}$$

if

1. $\Phi \neq \emptyset$ or $C \neq \bot$, and
2. there is a simultaneous mgu σ of C against Λ such that Λ contains the complement of each literal of $C\sigma$

Λ: $\quad P(u,u) \qquad Q(a,b)$

$$\sigma = \{\, x \mapsto u,\ y \mapsto u,\ z \mapsto b \,\}$$

$C : \neg P(x,y) \vee \neg Q(a,z)$

$C\sigma : \neg P(x,x) \vee \neg Q(a,b)$

Derivation Rules – Close Example

$$\textbf{Close}\ \frac{\Lambda \ \vdash \ \Phi, C}{\Lambda \ \vdash \ \bot}$$

if

1. $\Phi \neq \emptyset$ or $C \neq \bot$, and
2. there is a simultaneous mgu σ of C against Λ such that Λ contains the complement of each literal of $C\sigma$

$$\Lambda: \quad \underline{P(u,u)} \qquad \underline{Q(a,b)}$$

$$C : \neg P(x,y) \vee \neg Q(a,z)$$

$$\sigma = \{\, x \mapsto u,\ y \mapsto u,\ z \mapsto b \,\}$$

$$C\sigma : \neg \underline{P(x,x)} \vee \neg \underline{Q(a,b)}$$

2. satisfied **2. satisfied**

Close is applicable

Derivation Rules – Simplification Rules (1)

Propositional level:

$$\textbf{Subsume} \quad \frac{\Lambda,\ L \quad \vdash \quad \Phi,\ L \vee C}{\Lambda,\ L \quad \vdash \quad \Phi}$$

First-order level $\approx$ unit subsumption:

- All variables in context literal L must be universally quantified
- Replace equality by matching

Derivation Rules – Simplification Rules (2)

Propositional level:

Resolve
$$\frac{\Lambda,\ L \quad \vdash \quad \Phi,\ \overline{L} \lor C}{\Lambda,\ L \quad \vdash \quad \Phi,\ C}$$

First-order level $\approx$ restricted unit resolution

- All variables in context literal L must be universally quantified
- Replace equality by unification
- The unifier must not modify C

Derivation Rules – Simplification Rules (3)

Compact $$\frac{\Lambda,\ K,\ L \quad \vdash \quad \Phi}{\Lambda,\ K \qquad \vdash \quad \Phi}$$

if

1. all variables in K are universally quantified
2. $K\sigma = L$, for some substitution σ

Derivations and Completeness

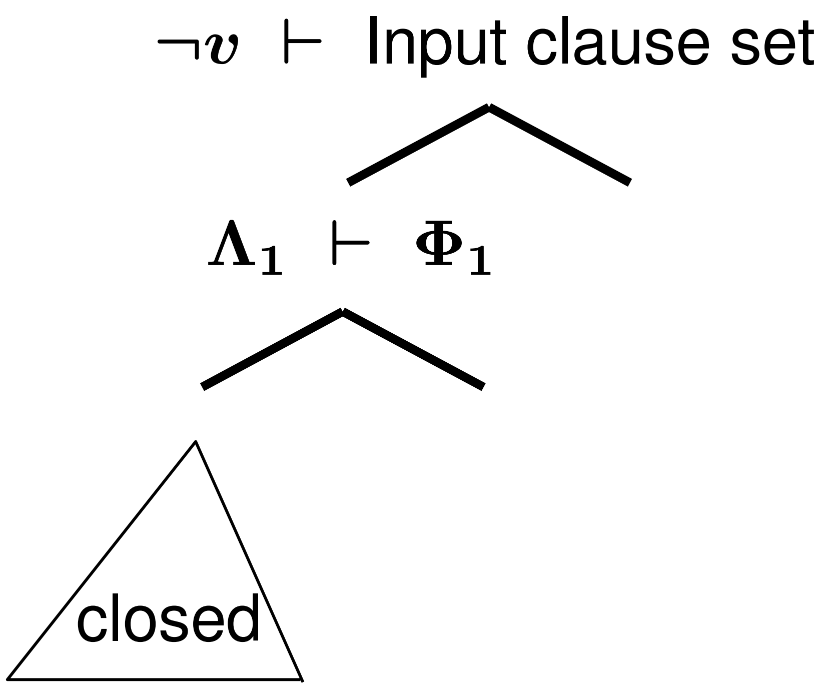
$\neg v \;\vdash\;$ Input clause set

Derivations and Completeness

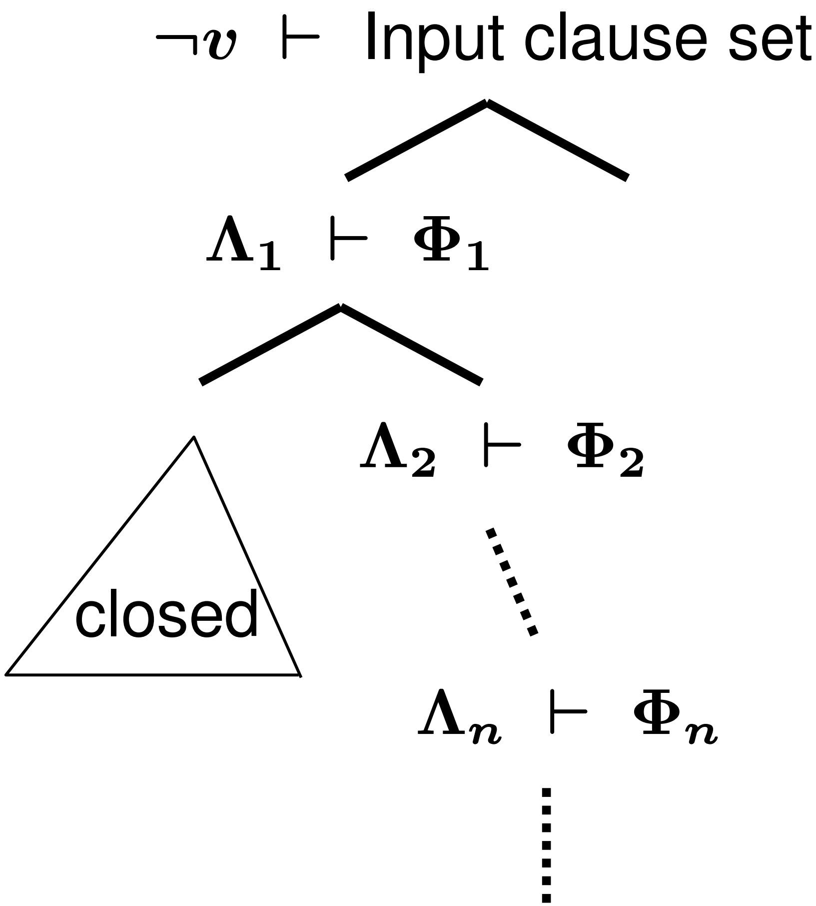
$\neg v \ \vdash$ Input clause set

$\Lambda_1 \ \vdash \ \Phi_1$

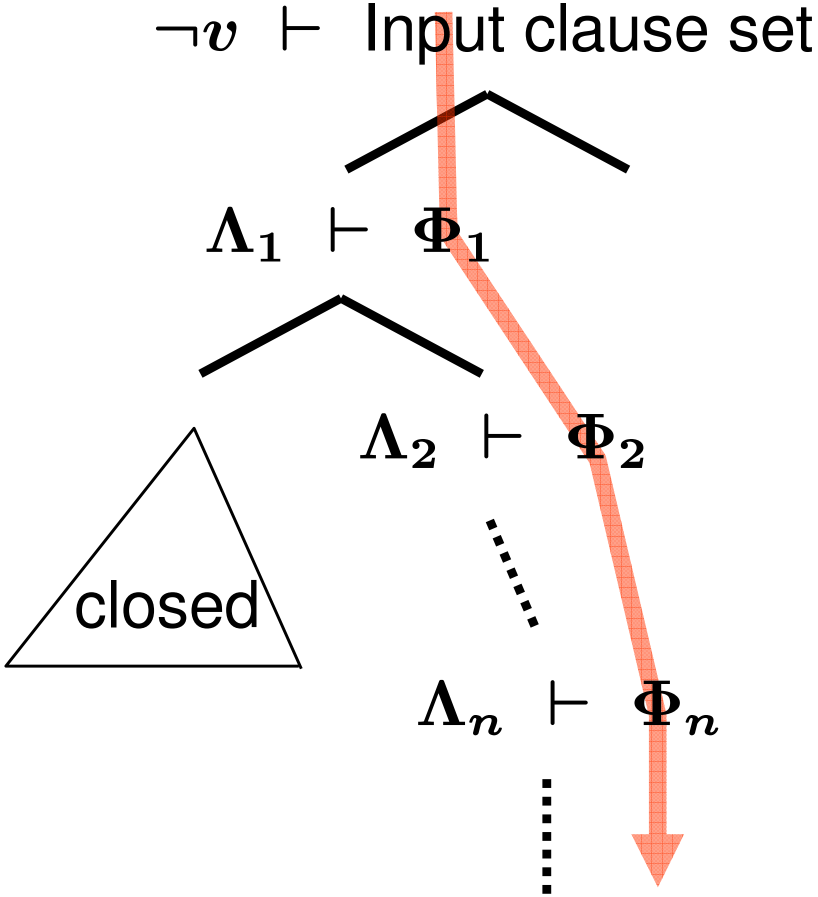
Derivations and Completeness



Derivations and Completeness

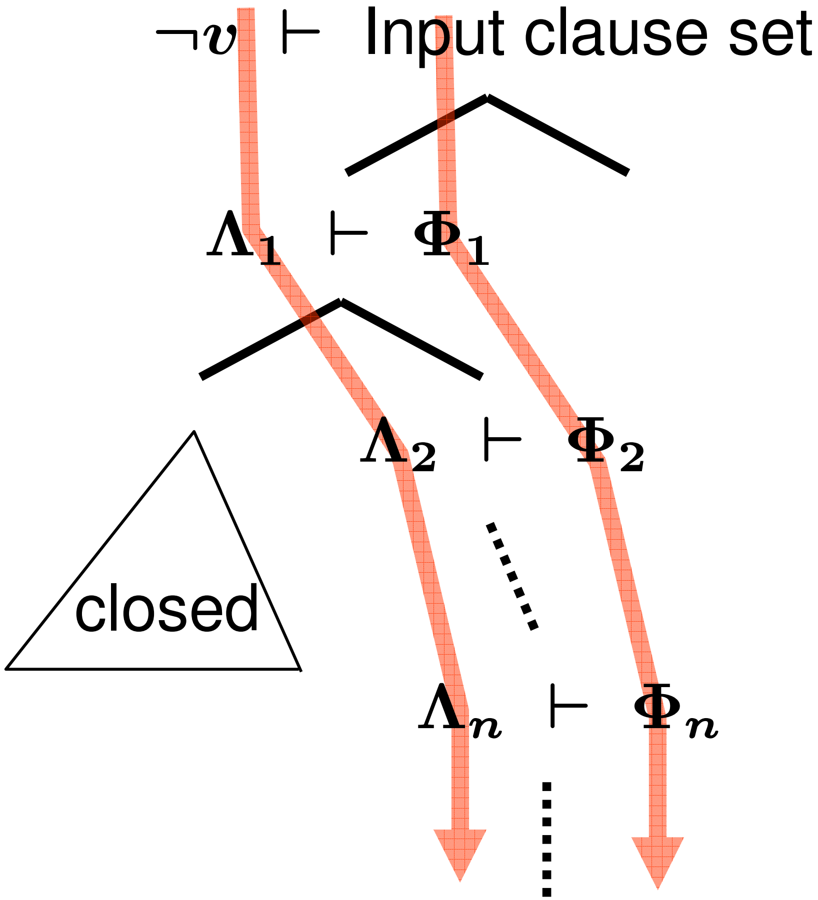


Derivations and Completeness



$$\Phi_\infty := \bigcup_{i \geq 0} \bigcap_{j \geq i} \Phi_j$$

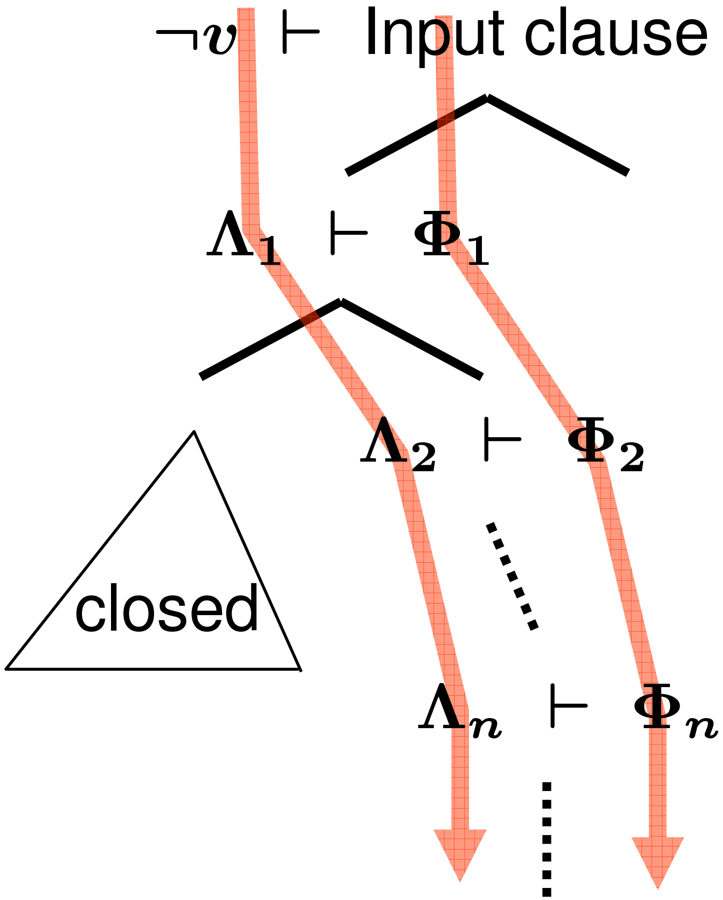
Derivations and Completeness



$$\Lambda_\infty := \bigcup_{i\geq 0} \bigcap_{j\geq i} \Lambda_j$$

$$\Phi_\infty := \bigcup_{i\geq 0} \bigcap_{j\geq i} \Phi_j$$

Derivations and Completeness



$$\Lambda_\infty := \bigcup_{i \geq 0} \bigcap_{j \geq i} \Lambda_j$$

$$\Phi_\infty := \bigcup_{i \geq 0} \bigcap_{j \geq i} \Phi_j$$

Fairness

Closed tree or open limit tree,
with some branch satisfying:

1. Close not applicable to any Λ_i
2. For all $C \in \Phi_\infty$ and subst. γ,
   "if for some i, $\Lambda_i \not\models C\gamma$
   then there is $j \geq i$
   such that $\Lambda_j \models C\gamma$

(Use Split to achieve this)

Completeness

Suppose a fair derivation
of an open limit tree

Show that $\Lambda_\infty \models \Phi_\infty$

Implementation: Darwin

- „Serious“ Implementation

  Part of Master Thesis, continued in Ph.D. project (A. Fuchs)
- (Intended) Applications
  - detecting dependent variables in CSP problems
  - strong equivalence of logic programs
  - Finite countermodels for program verification purposes
  - Bernays-Schoenfinkel fragment of autoepistemic logic
- Currently extended:
  - Lemma learning
  - Equality inference rules [Baumgartner and Tinelli, 2005]
- Written in OCaml, 14K LOC
- User manual, proof tree output (GraphViz)
- Download at `http://goedel.cs.uiowa.edu/Darwin/`

FDPLL/ME vs. OSHL

Recall OSHL:

- Stepwisely 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 $\perp$-version $S_\perp$ 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:**

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

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

- 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: $\forall x (P(x) \vee Q(x))$
$\neg P(a)$
$\neg P(b)$
$\neg Q(b)$

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

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

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

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

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

- **Connection method (and tableaux) are 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:

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

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

- \- 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)

Open Research Problem

- **ARM (atomic representation of models) [Gottlob and Pichler, 1998]**
  ARM: set of atoms. Set of all ground instances is an interpretation
- **Contexts are stronger than ARMs. E.g., for $\Lambda = \{P(u,v), \neg P(u,u)\}$ and $\Sigma_F = \{a/0, f/1\}$ there is no equivalent ARM**
- **Contexts are equivalent to DIGs (Disjunctions of Implicit Generalizations) [Fermüller and Pichler, 2005]**
- **Contexts cannot represent certain infinite interpretations, e.g. minimal models of the clause set**

$$P(x) \vee P(f(x)),\ \neg P(x) \vee \neg P(f(x))$$

Open Research Problem

- **ARM (atomic representation of models) [Gottlob and Pichler, 1998]**
  ARM: set of atoms. Set of all ground instances is an interpretation
- **Contexts are stronger than ARMs. E.g., for** $\Lambda = \{P(u,v), \neg P(u,u)\}$ **and** $\Sigma_F = \{a/0, f/1\}$ **there is no equivalent ARM**
- **Contexts are equivalent to DIGs (Disjunctions of Implicit Generalizations) [Fermüller and Pichler, 2005]**
- **Contexts cannot represent certain infinite interpretations, e.g. minimal models of the clause set**

$$P(x) \vee P(f(x)),\ \neg P(x) \vee \neg P(f(x))$$

Instance Based Method based on more powerful model representation?

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)

- **Singular inference rule: Linking**

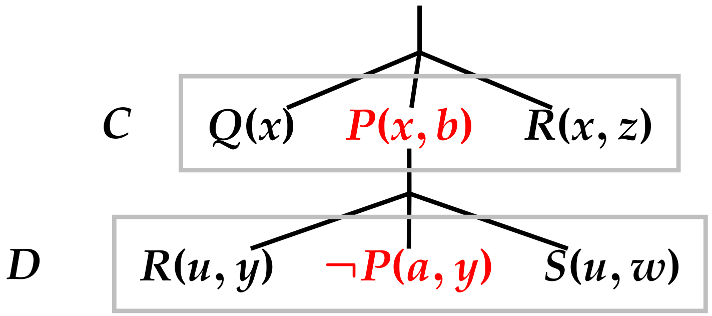
C $\boxed{Q(x) \quad P(x,b) \quad R(x,z)}$

D $\boxed{R(u,y) \quad \neg P(a,y) \quad S(u,w)}$

potentially complementary literals on path

The Disconnection Calculus (II)

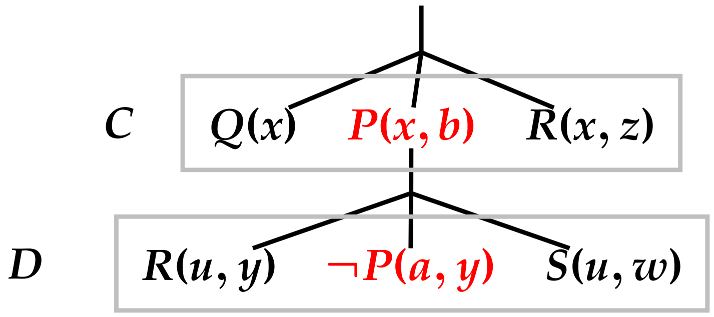
- **Singular inference rule: Linking**

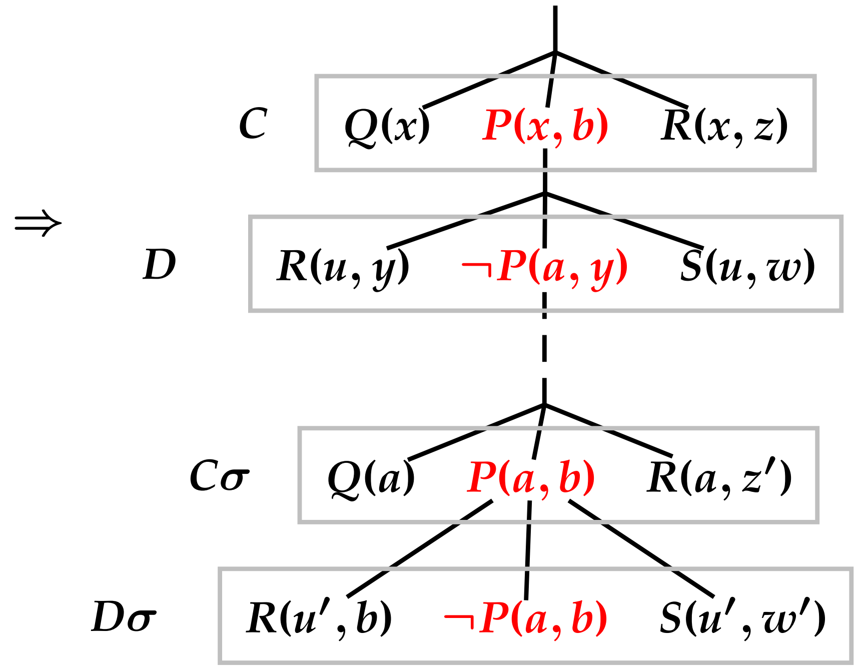


unifier for literals:
$\{x/a, y/b\}$

The Disconnection Calculus (II)

- **Singular inference rule: Linking**

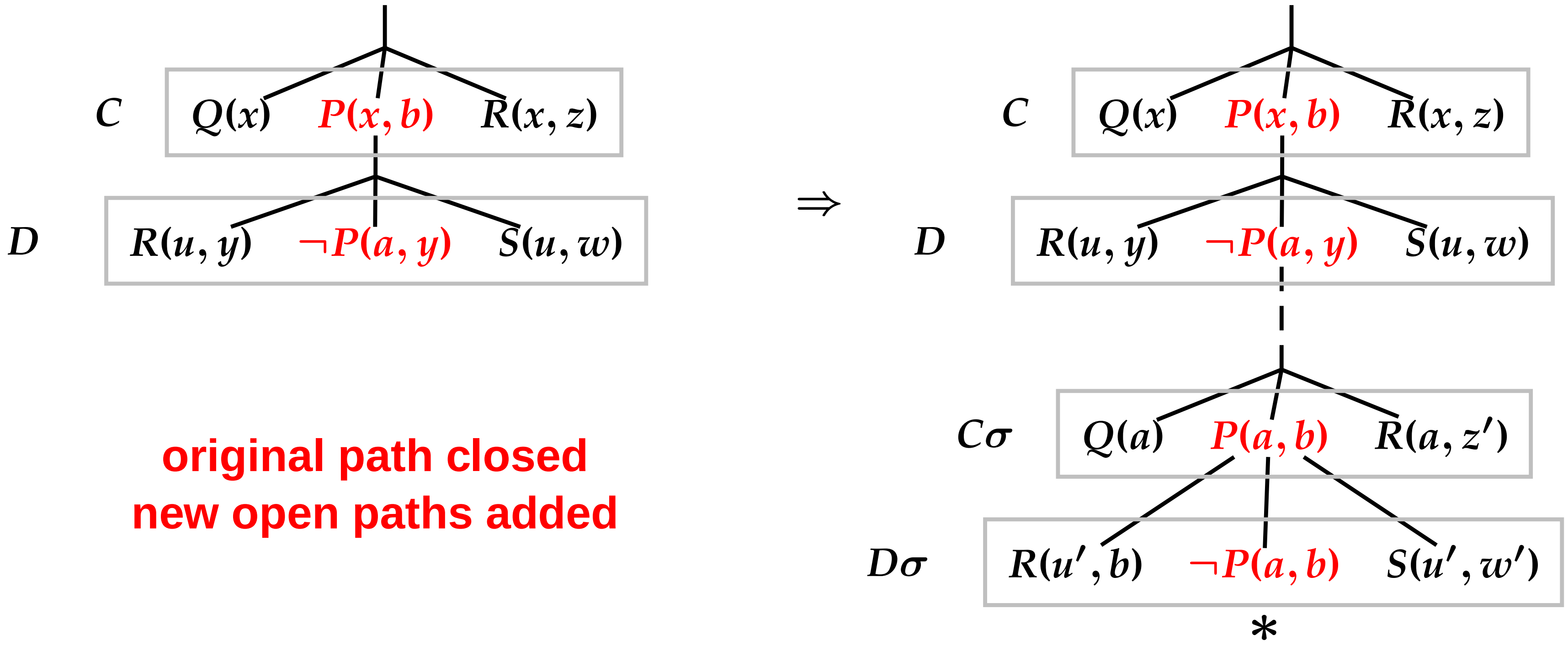




append instances with substitution $\{x/a, y/b\}$ to path

The Disconnection Calculus (II)

- Singular inference 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

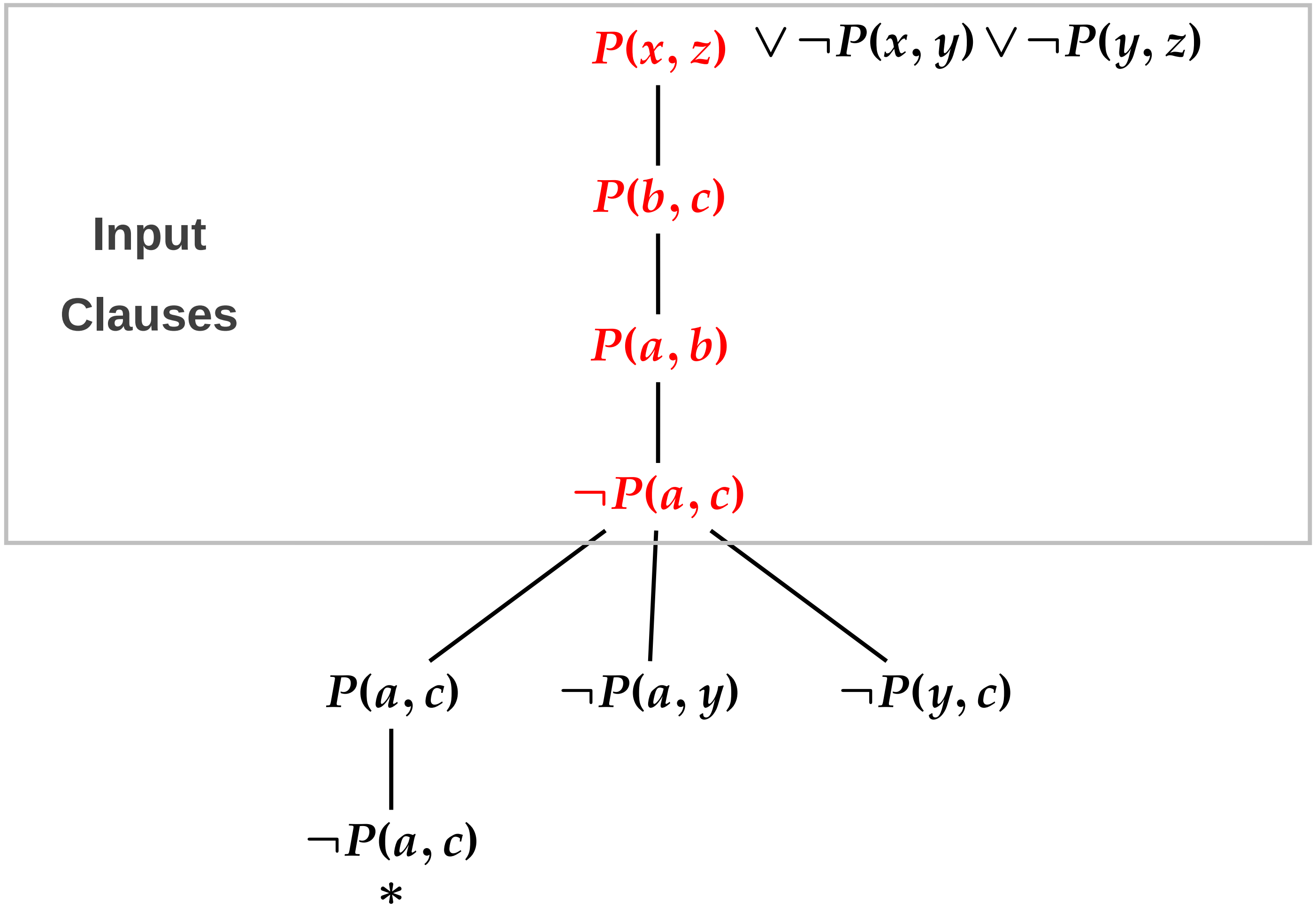
$P(x, z) \vee \neg P(x, y) \vee \neg P(y, z)$

$P(b, c)$

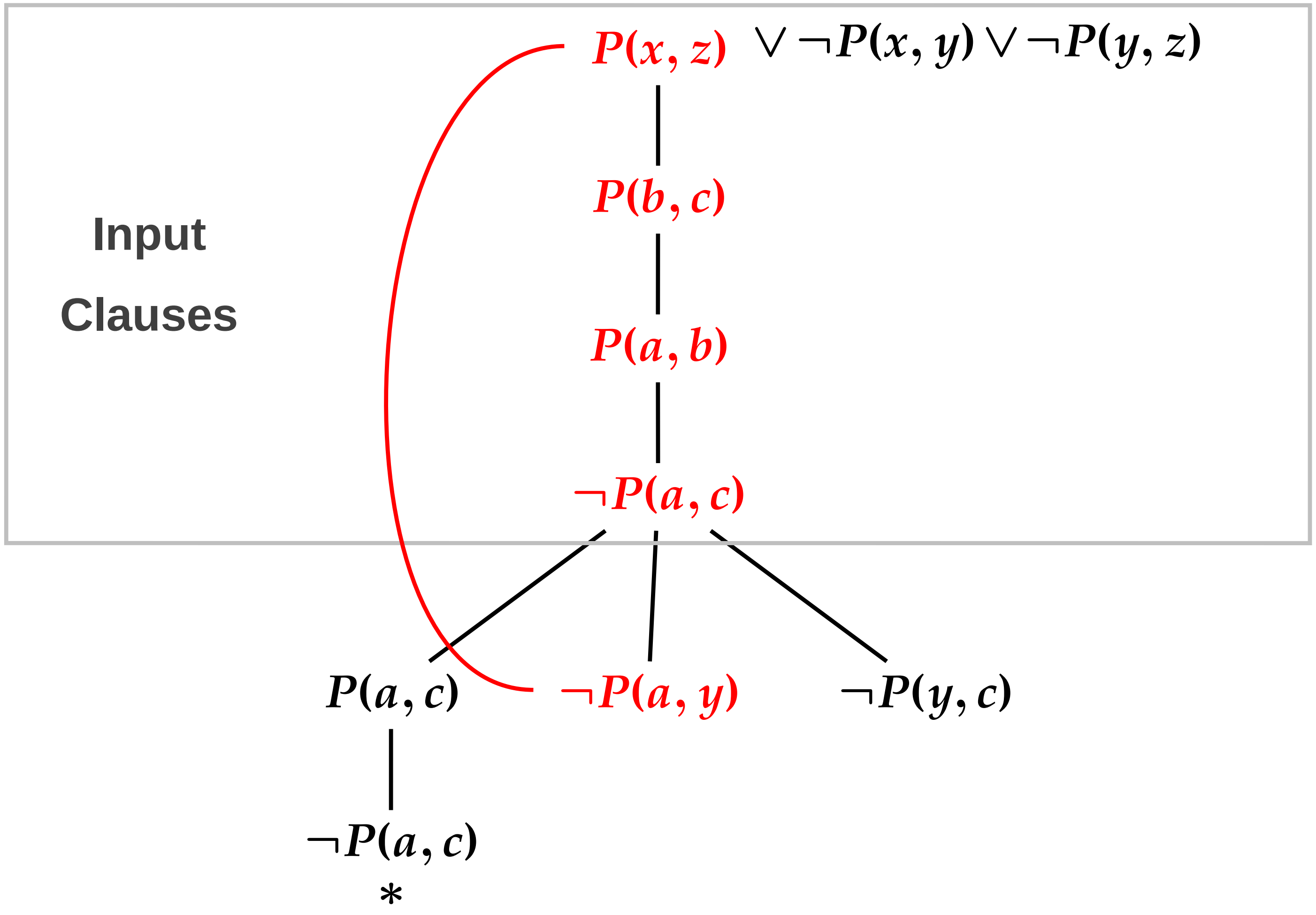
$P(a, b)$

$\neg P(a, c)$

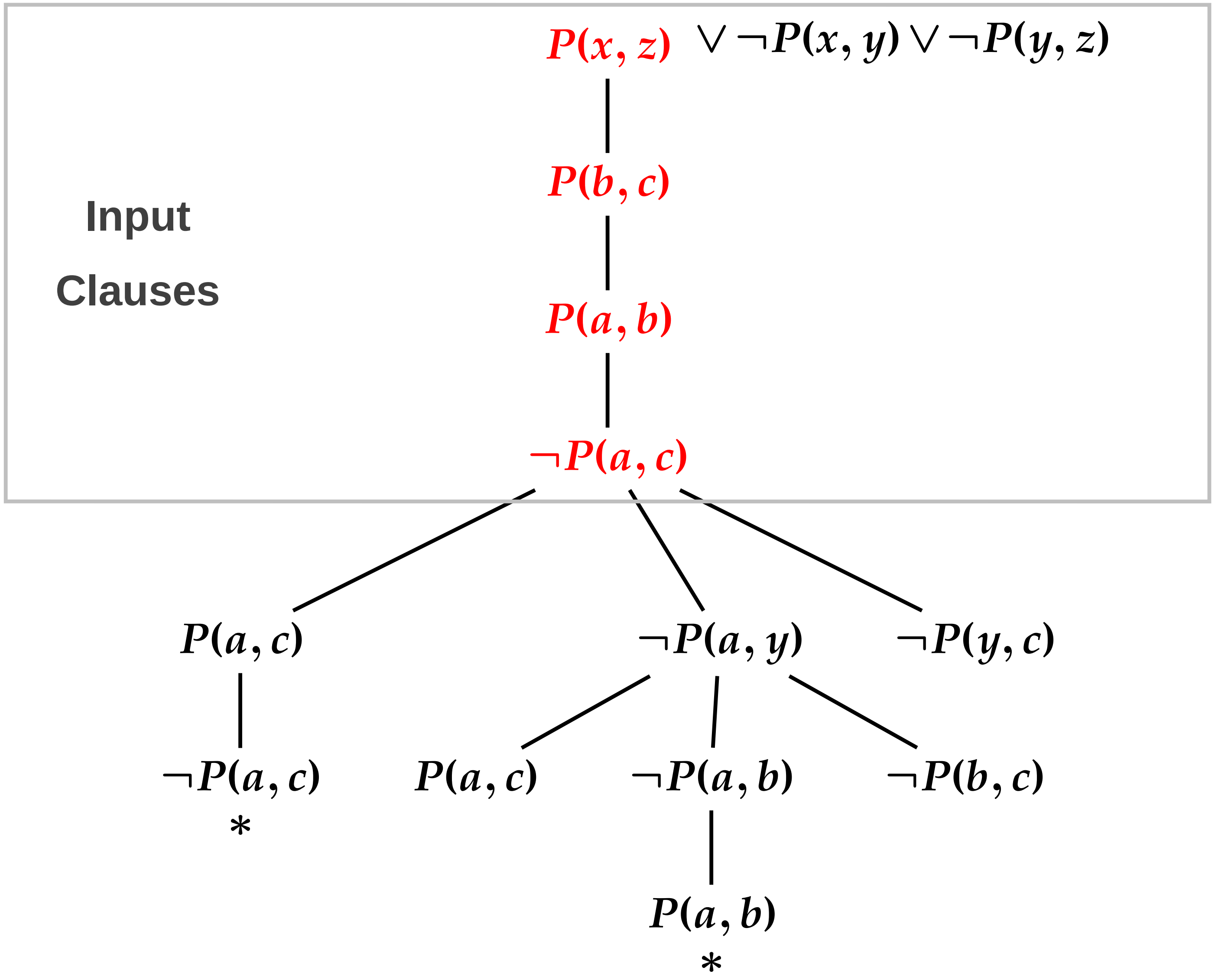
An Example Proof



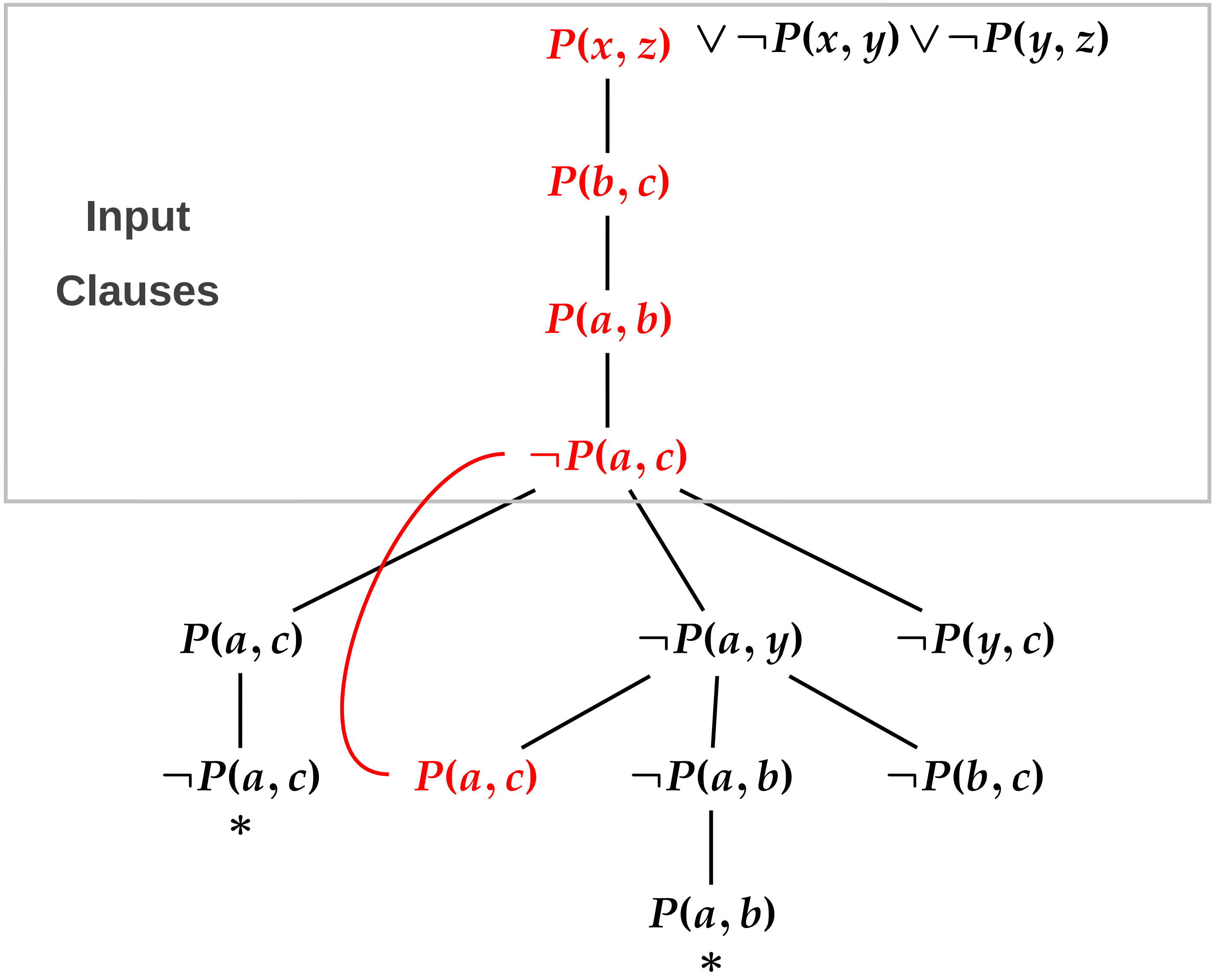
An Example Proof



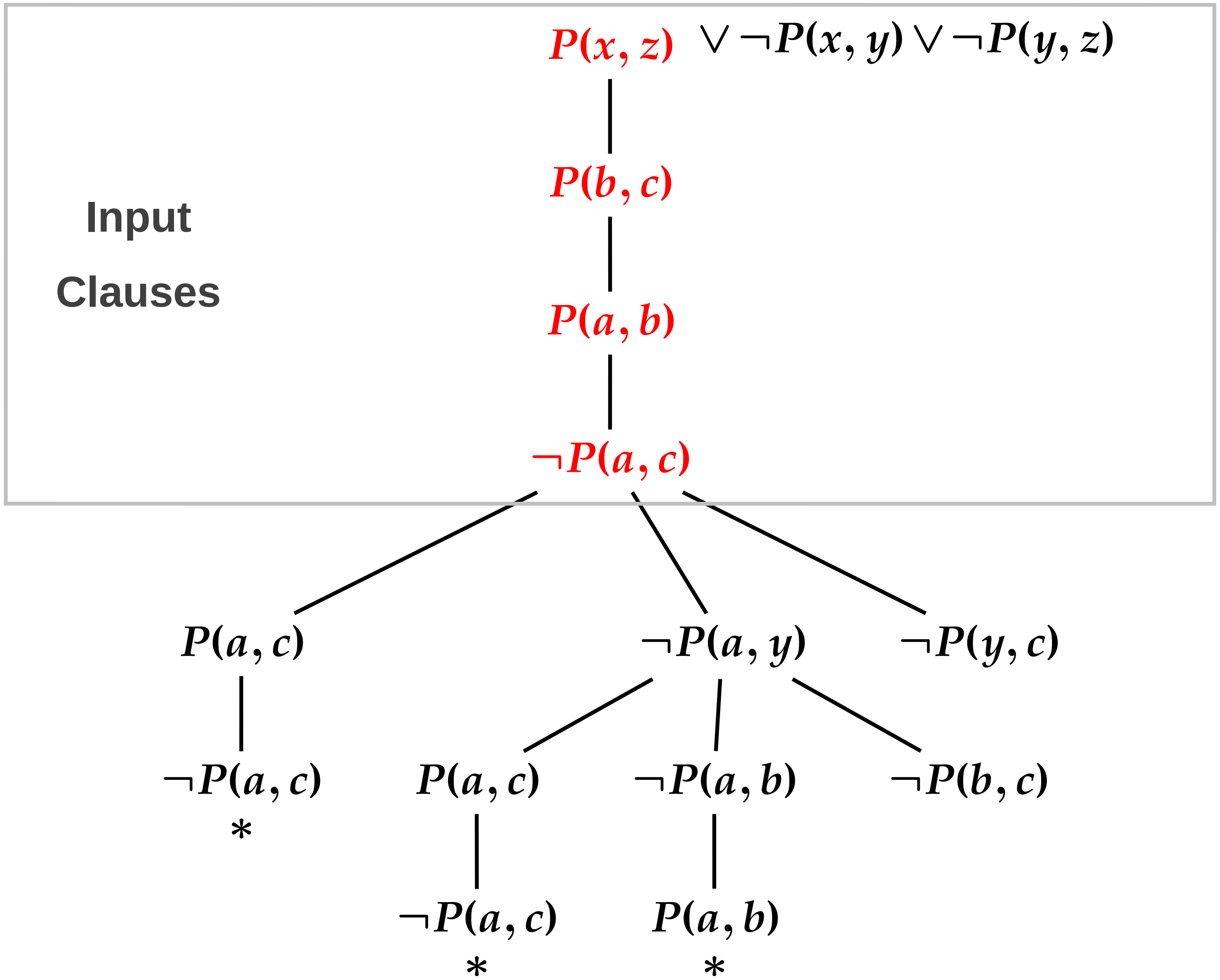
An Example Proof



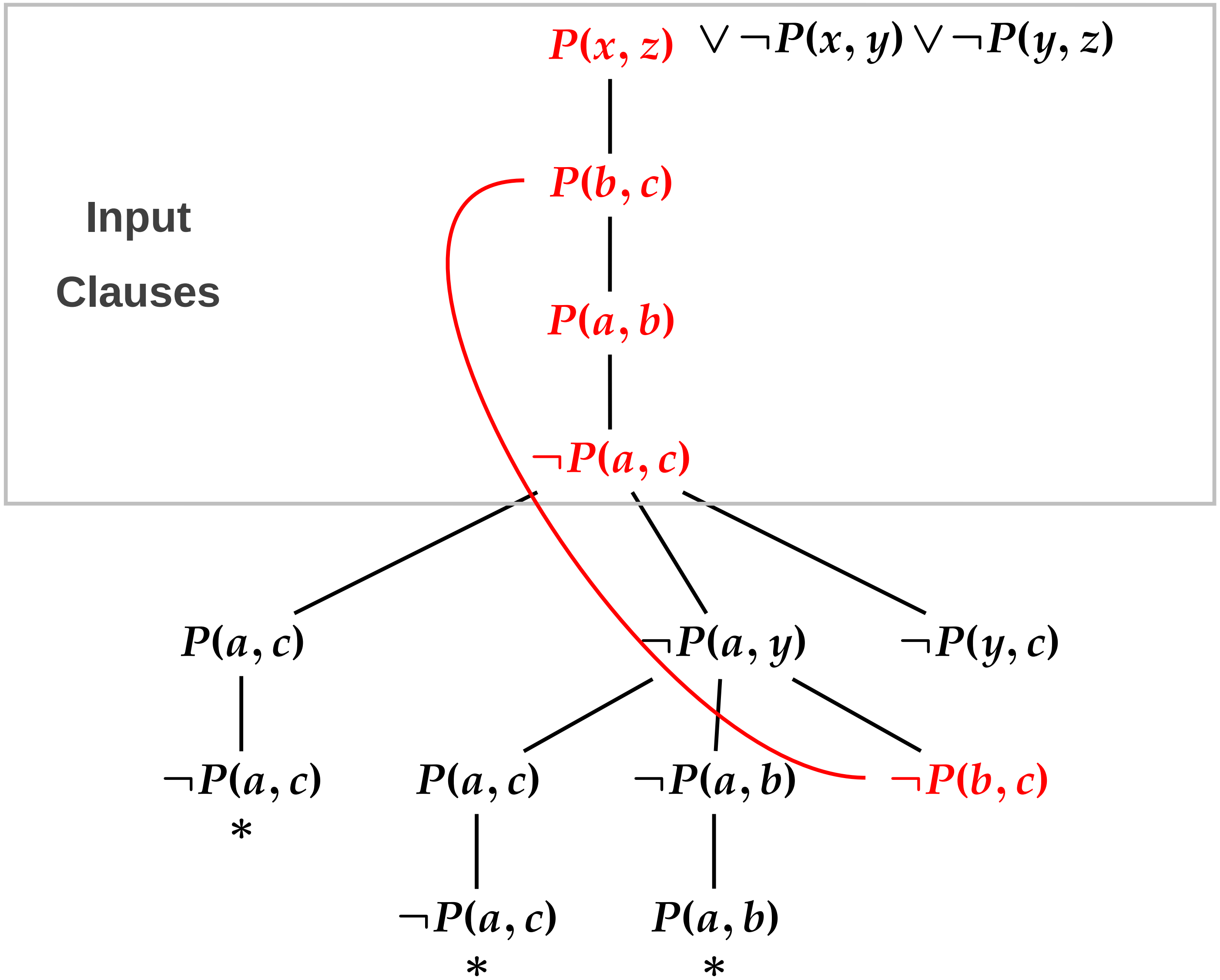
An Example Proof



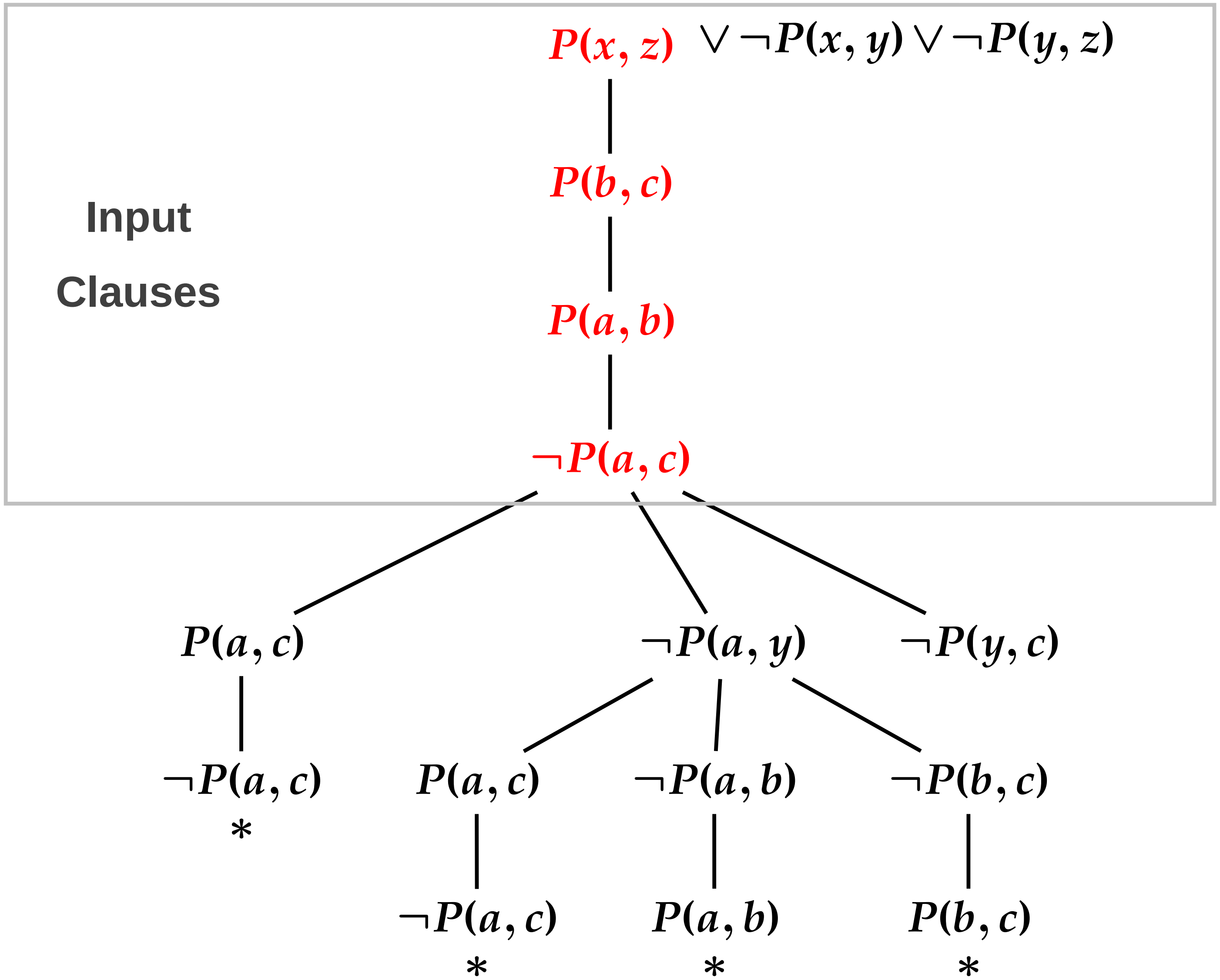
An Example Proof



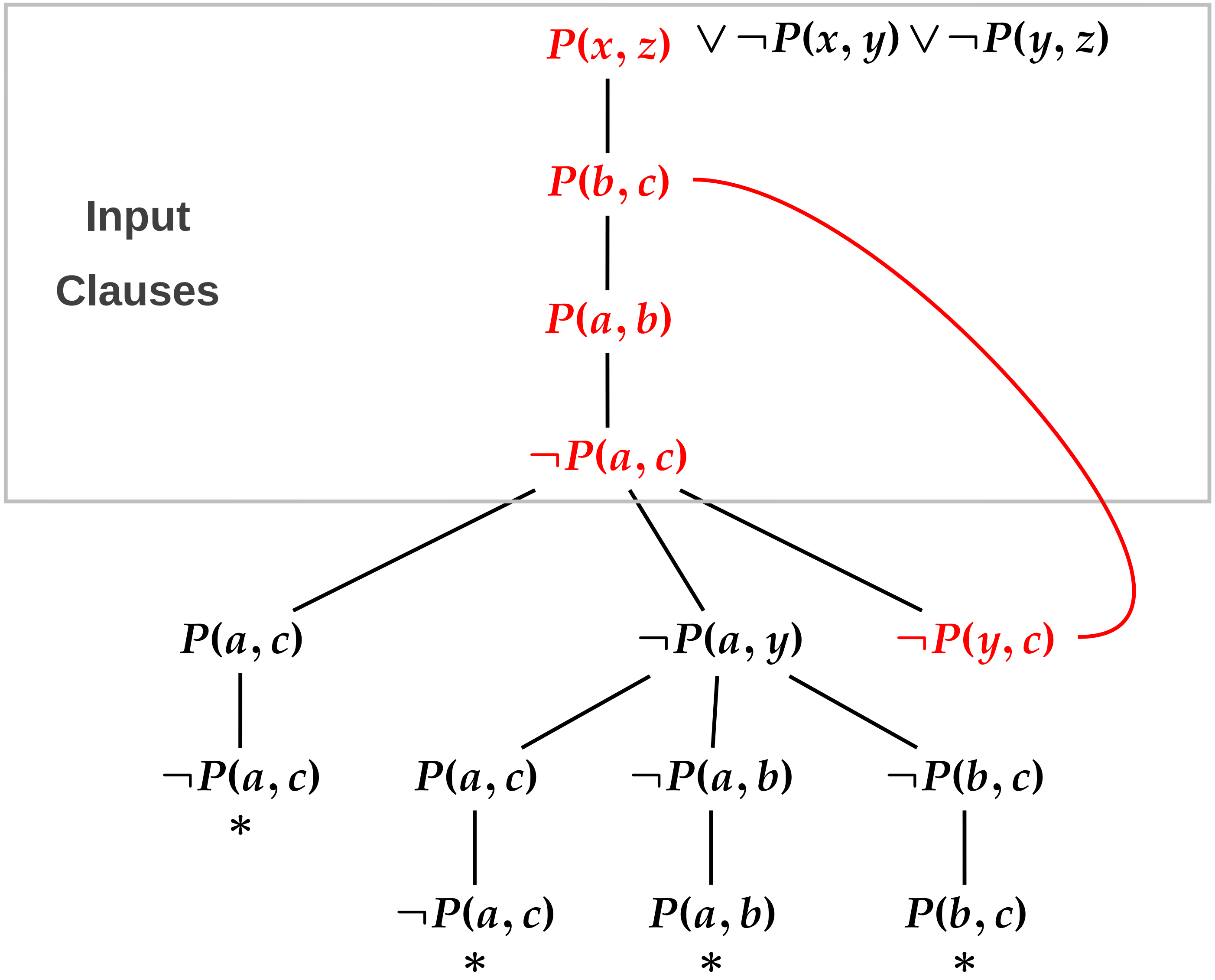
An Example Proof



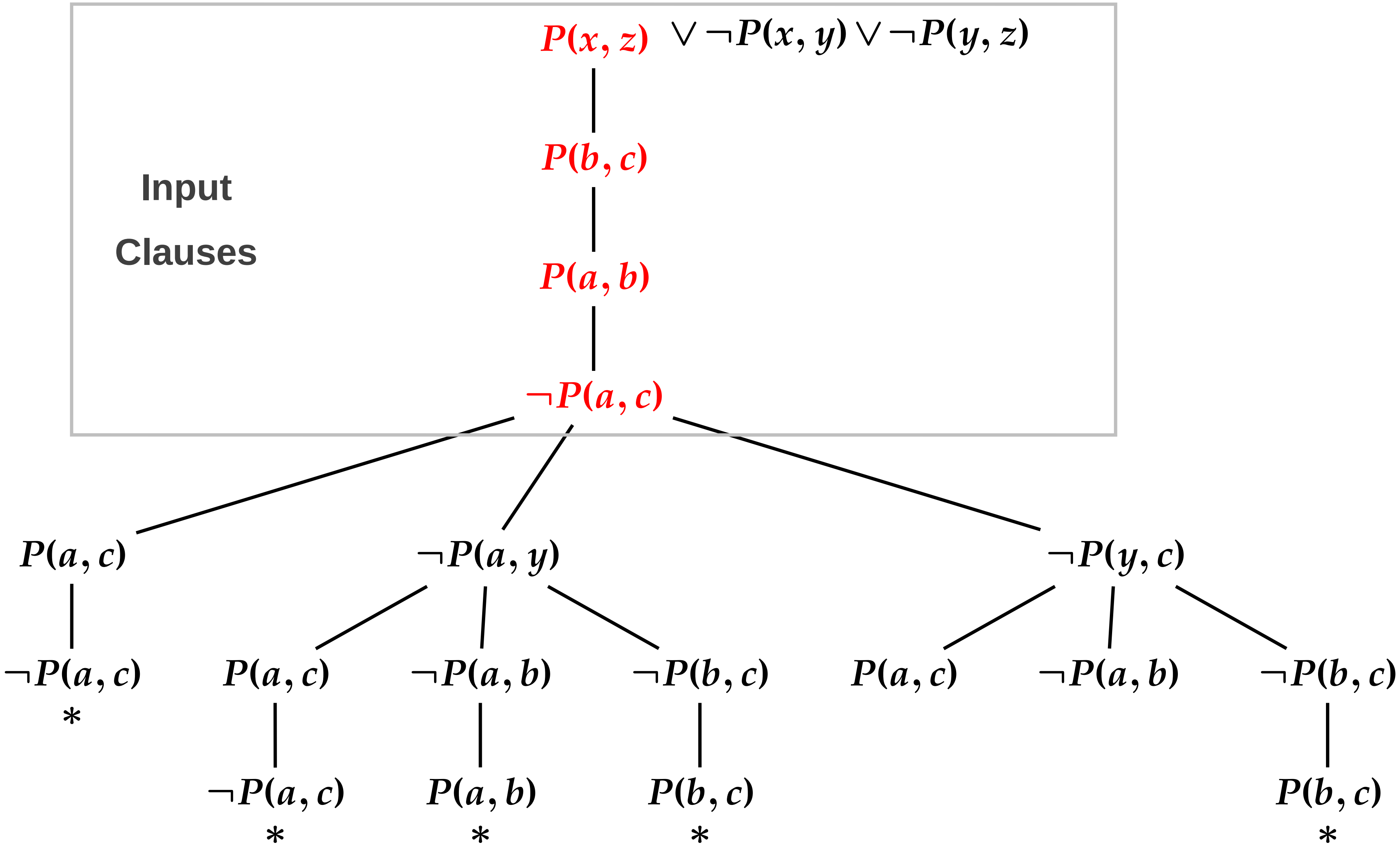
An Example Proof



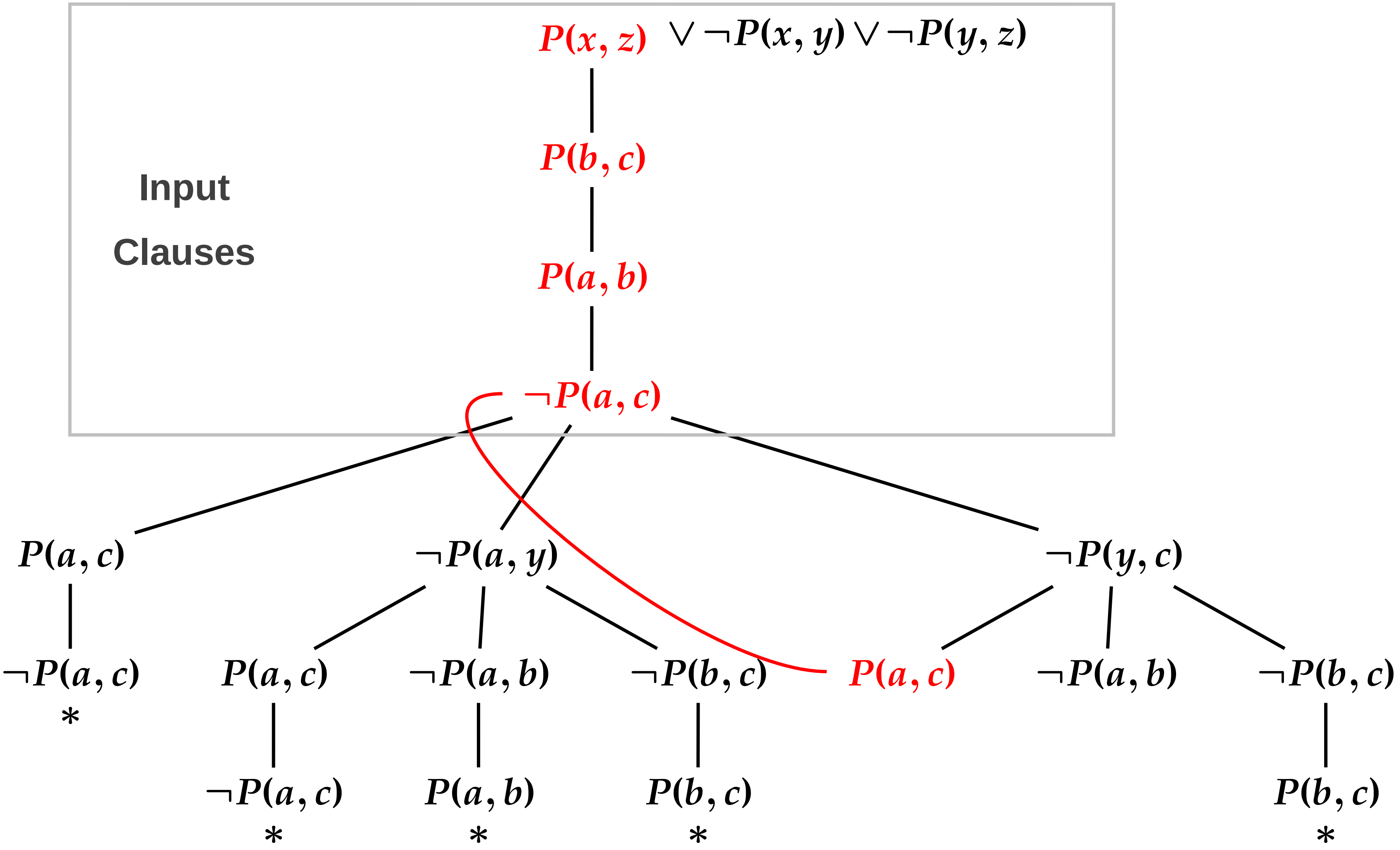
An Example Proof



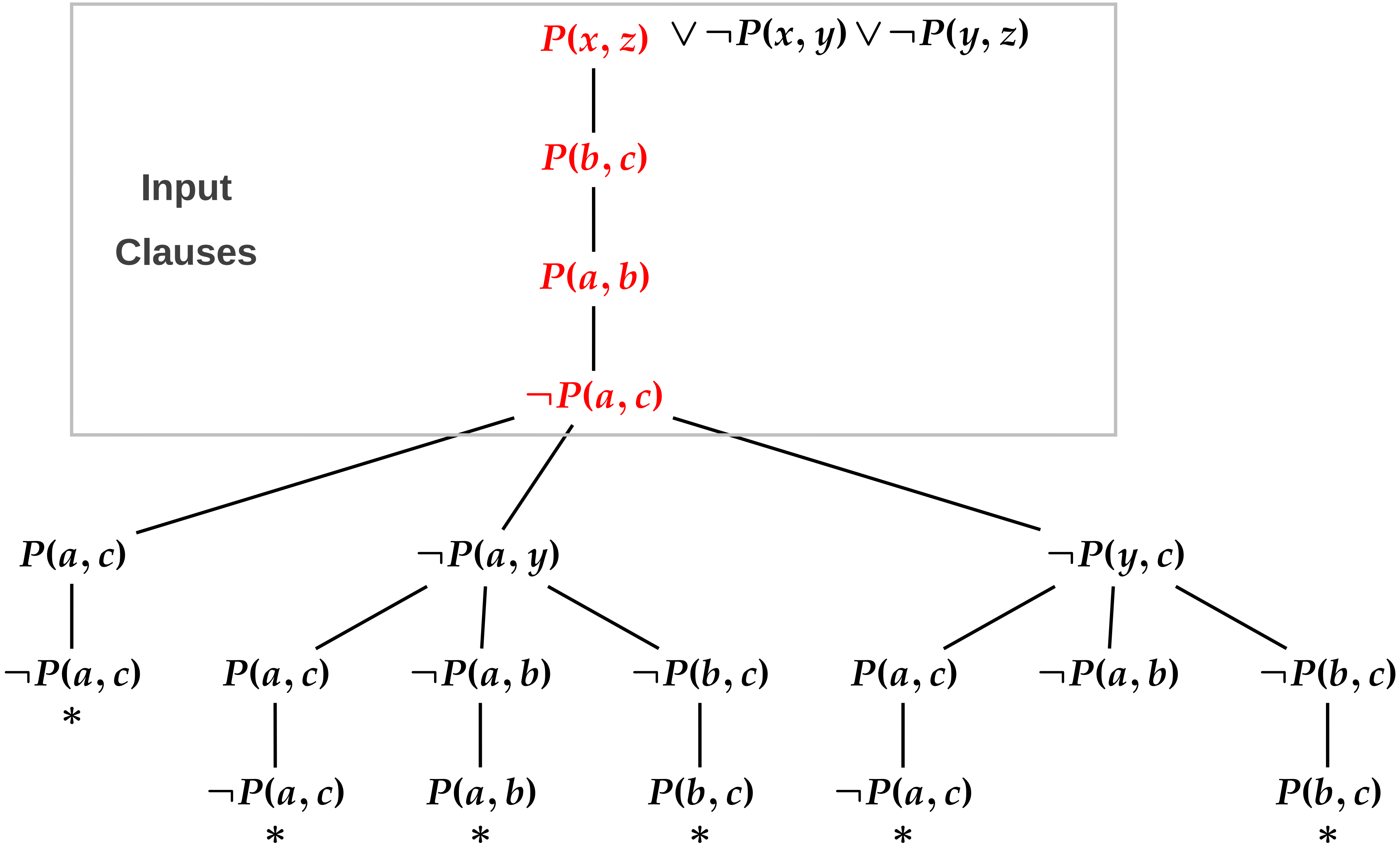
An Example Proof



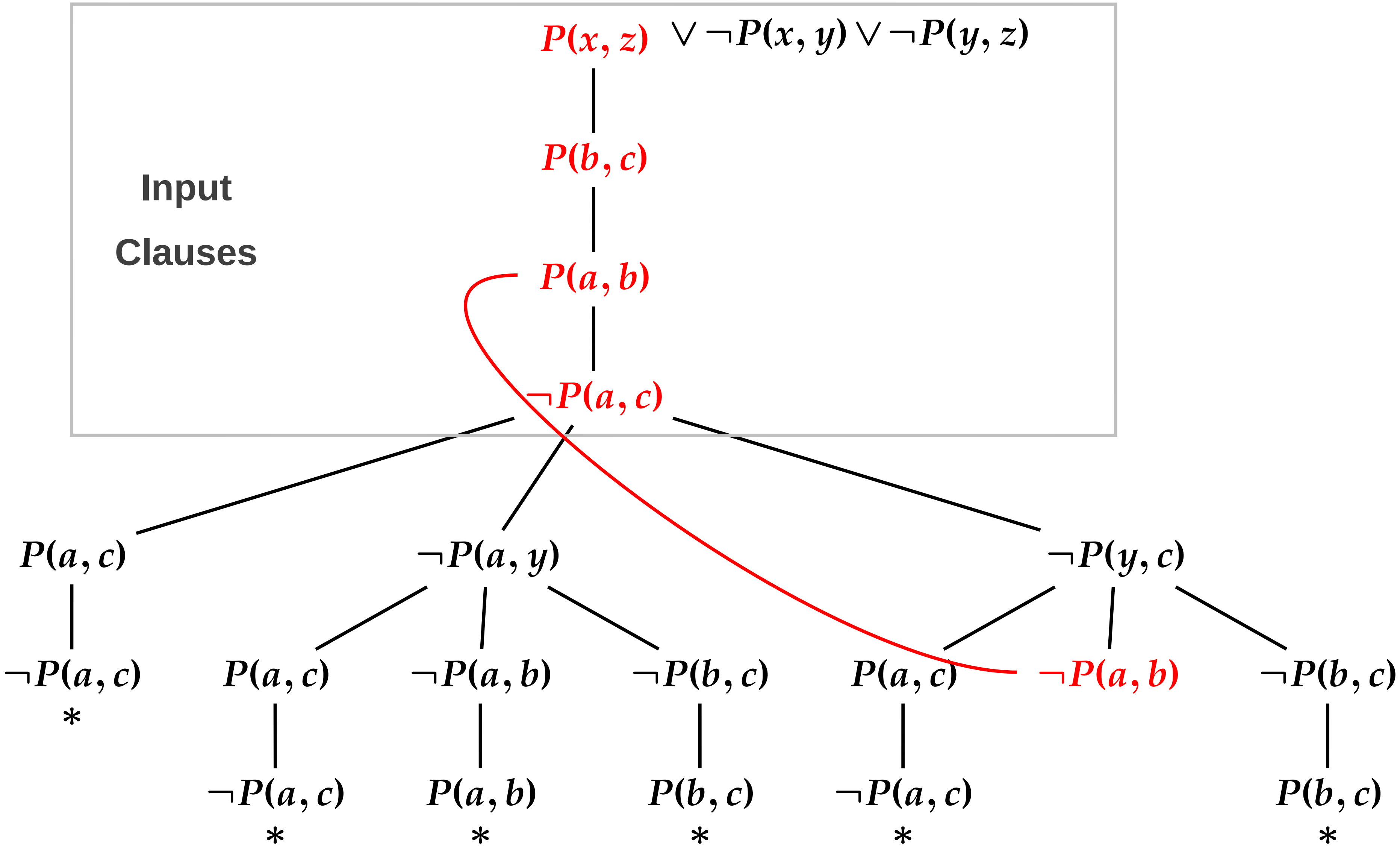
An Example Proof



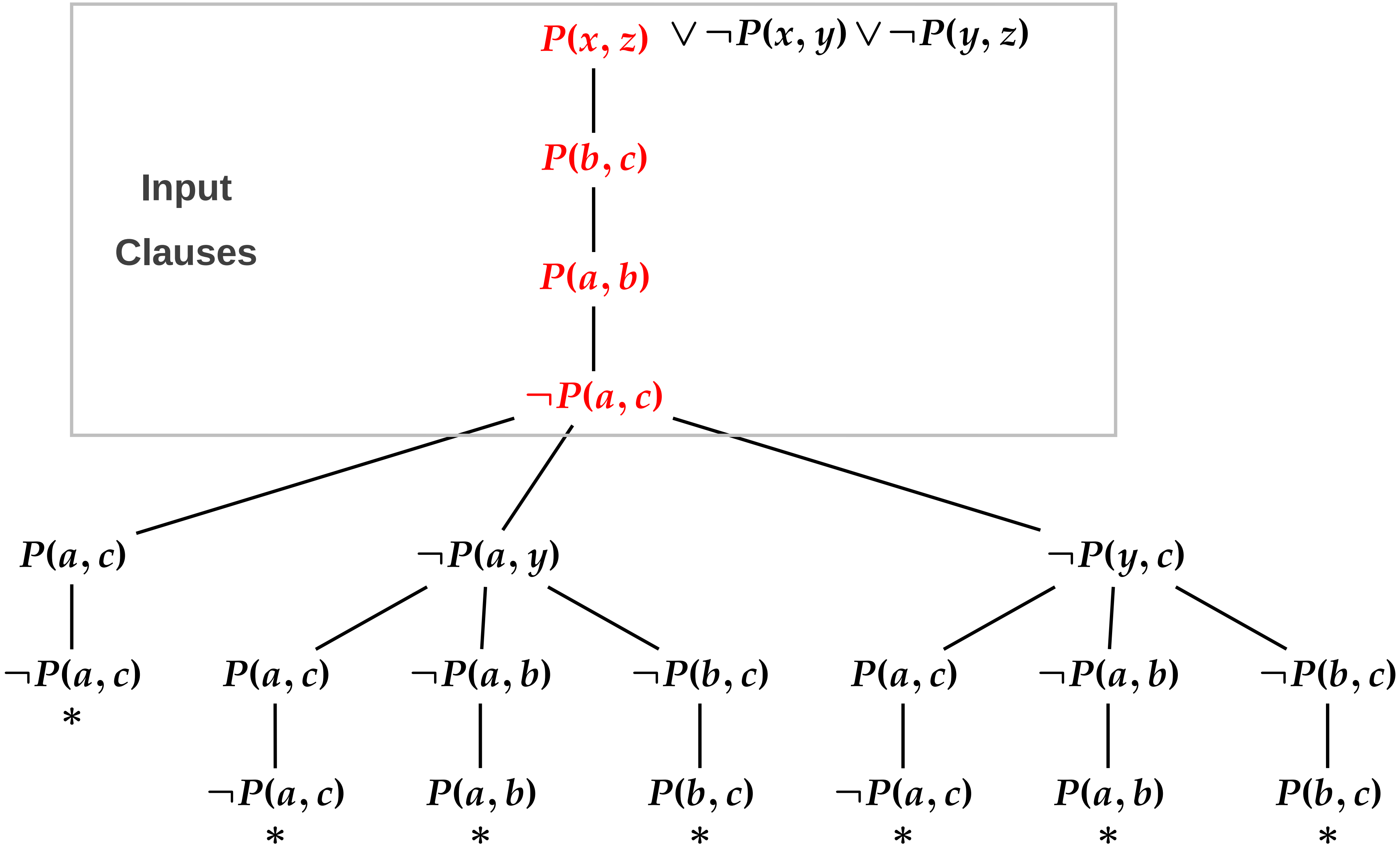
An Example Proof



An Example Proof



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

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

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

$$P(b,c)$$

$$P(a,b)$$

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

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

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

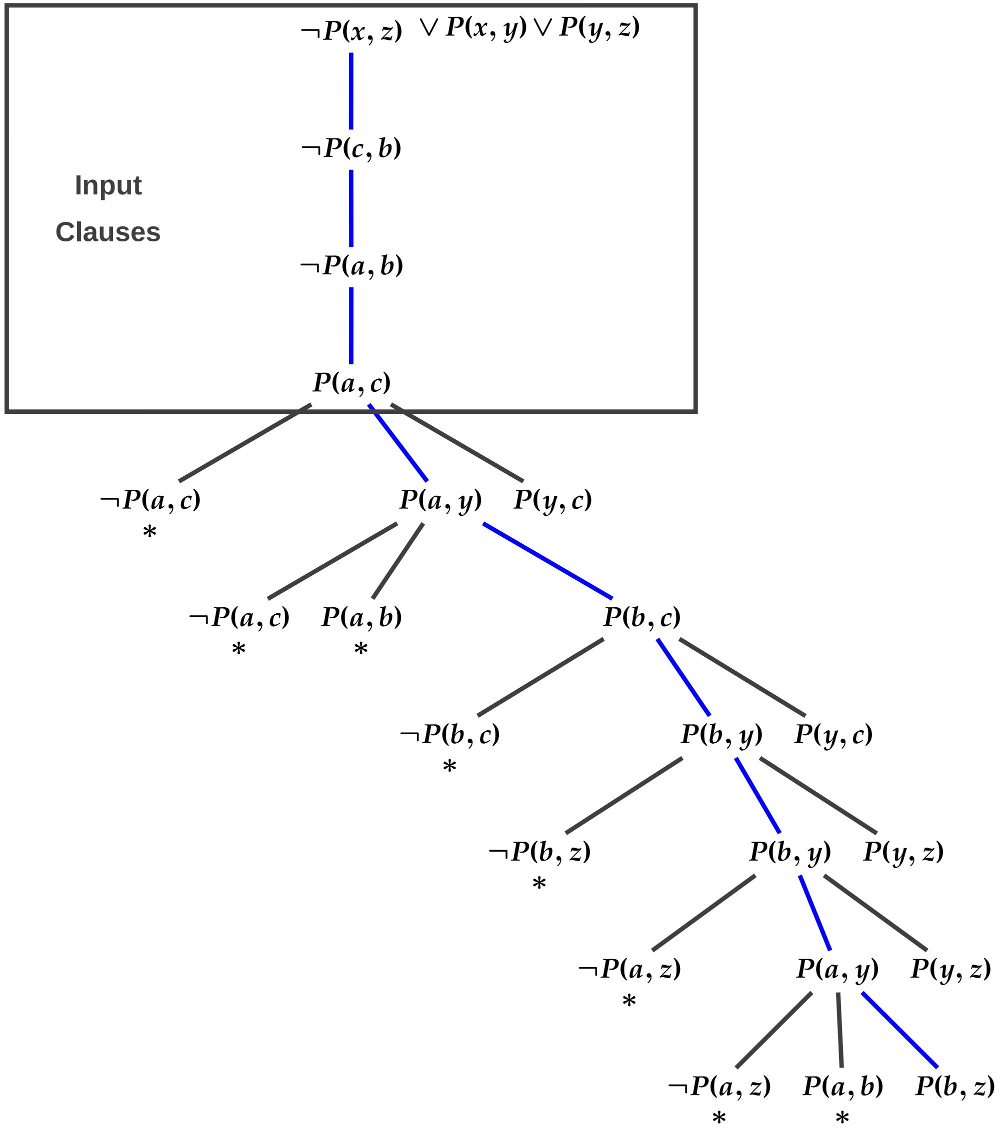
$$P(b,c)$$

$$P(a,b)$$

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

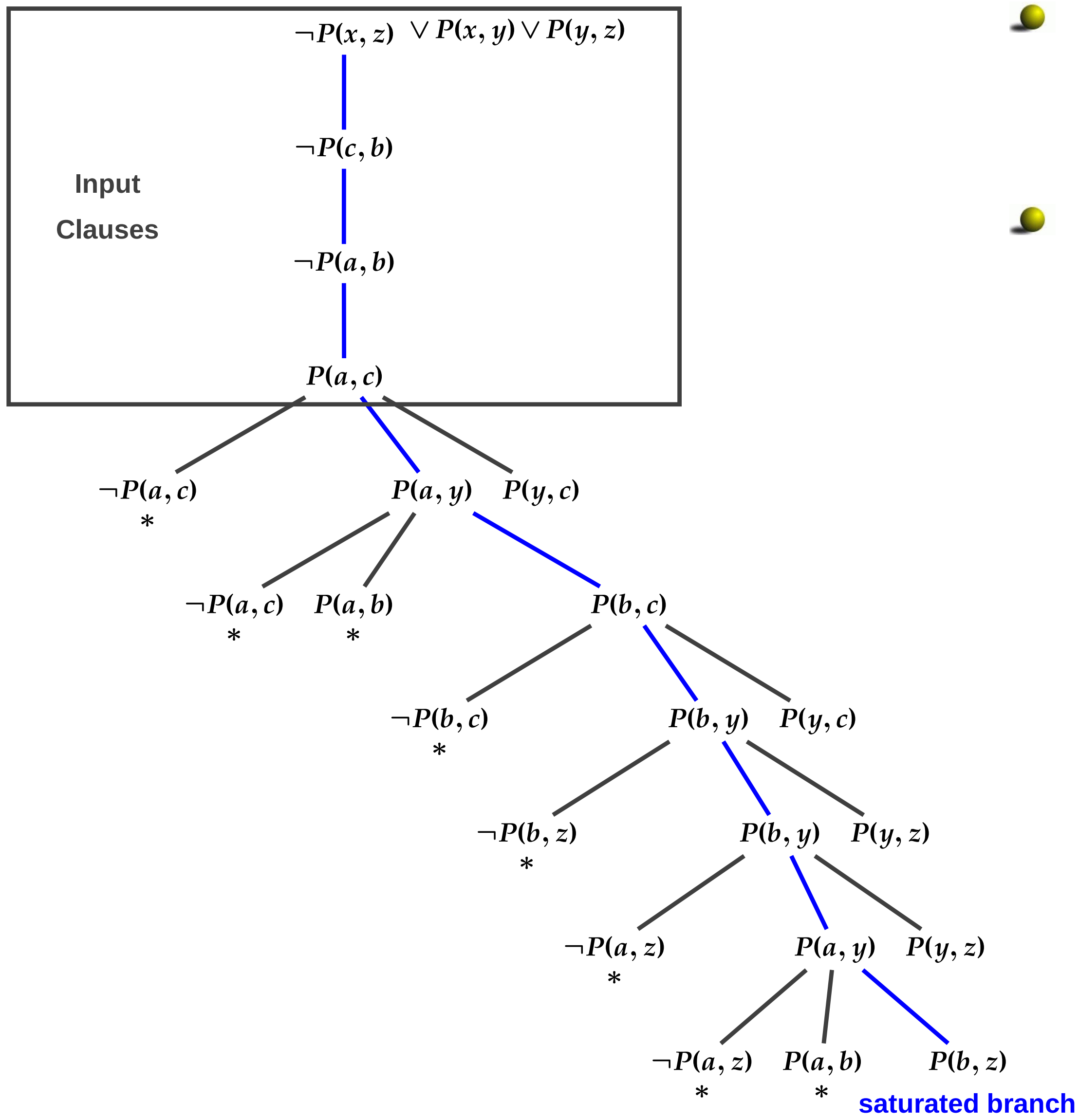
- **Again, we attempt to find a proof**

A Saturated Open Tableau



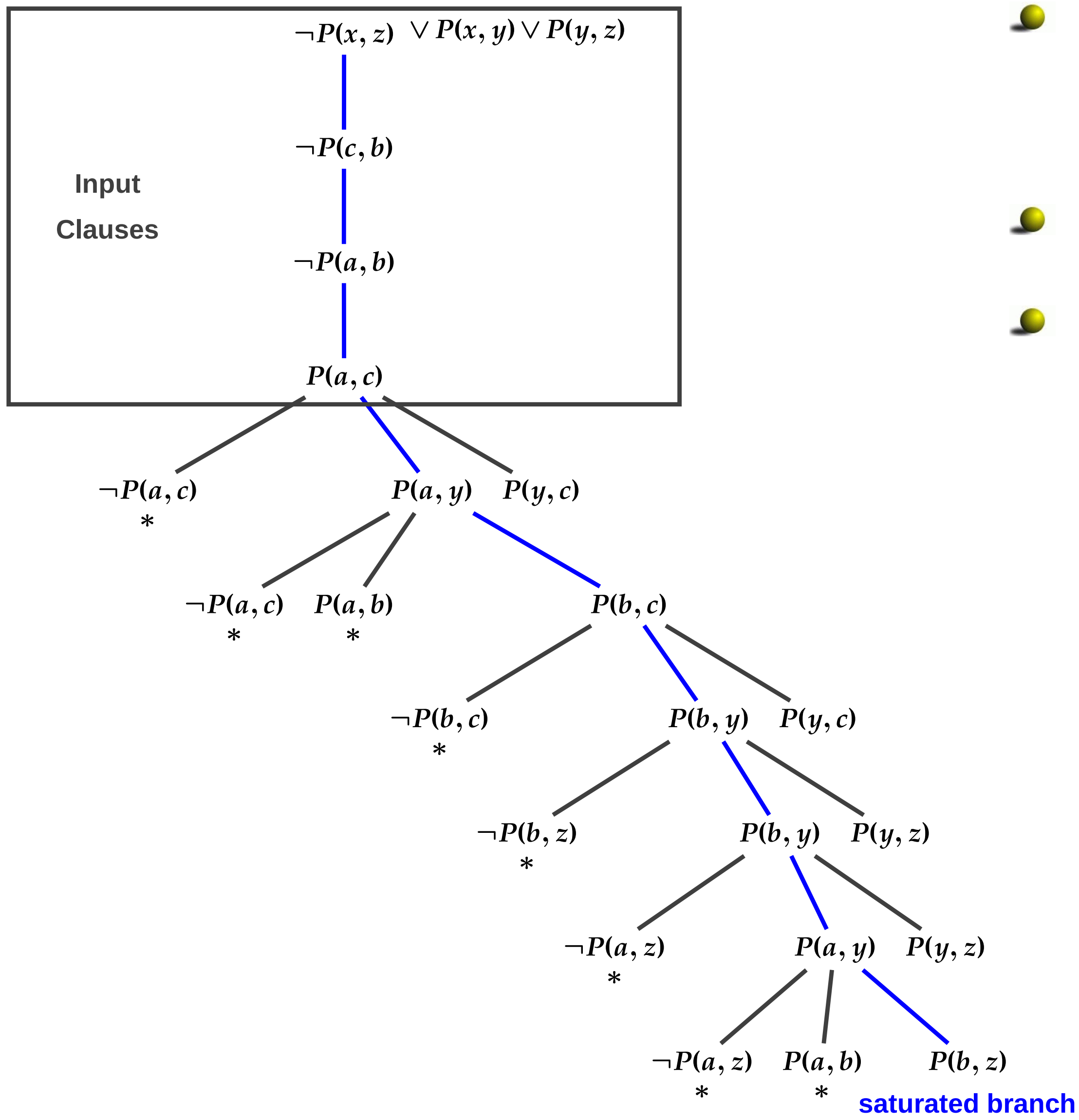
- **This open tableau cannot be closed**

A Saturated Open Tableau



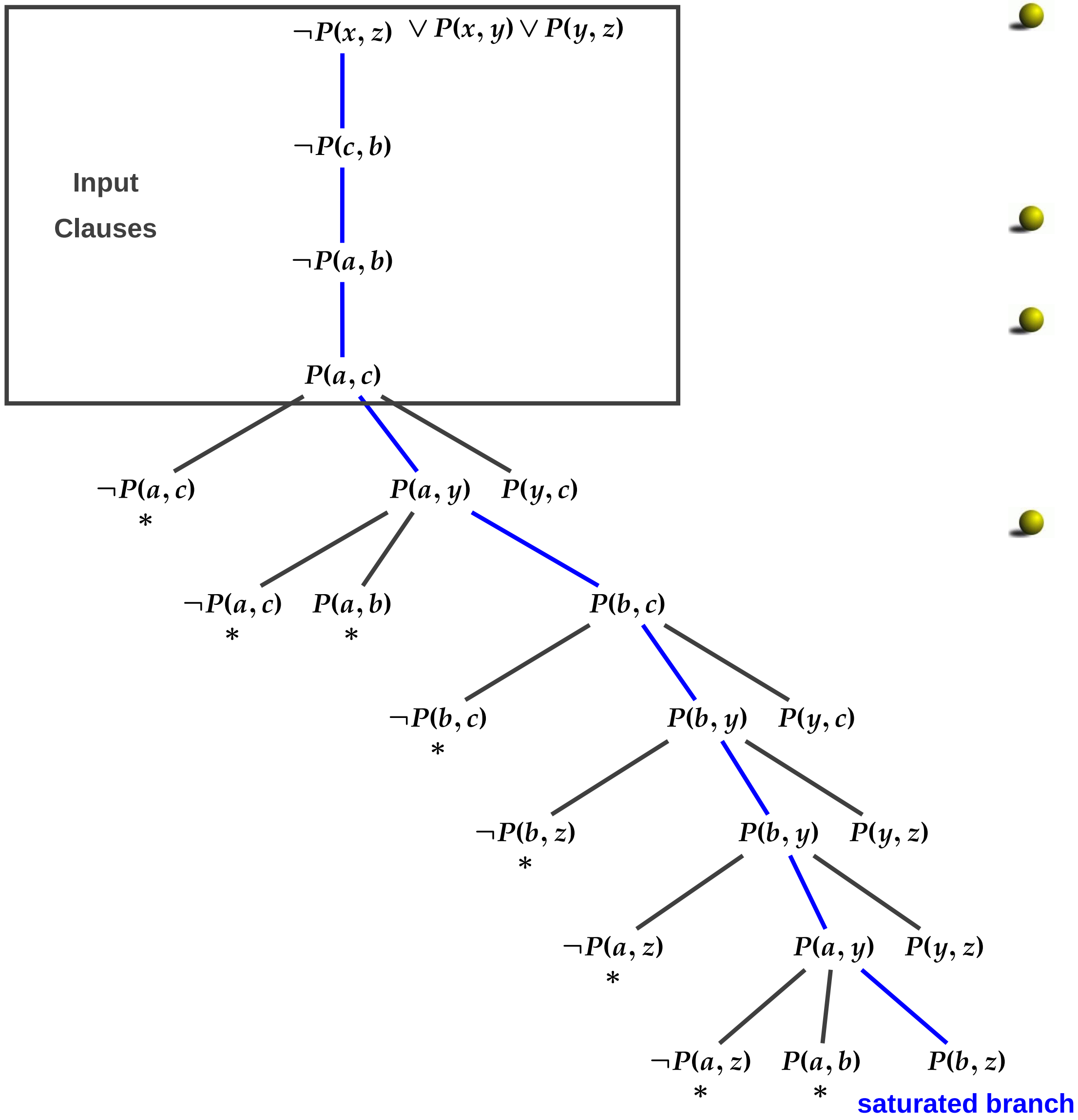
- **This open tableau cannot be closed**
- **Indicated branch is saturated**

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

- *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

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

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:

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

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

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

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

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

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

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

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

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

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

Model Extraction

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

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

From which we get the finite Herbrand model:

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

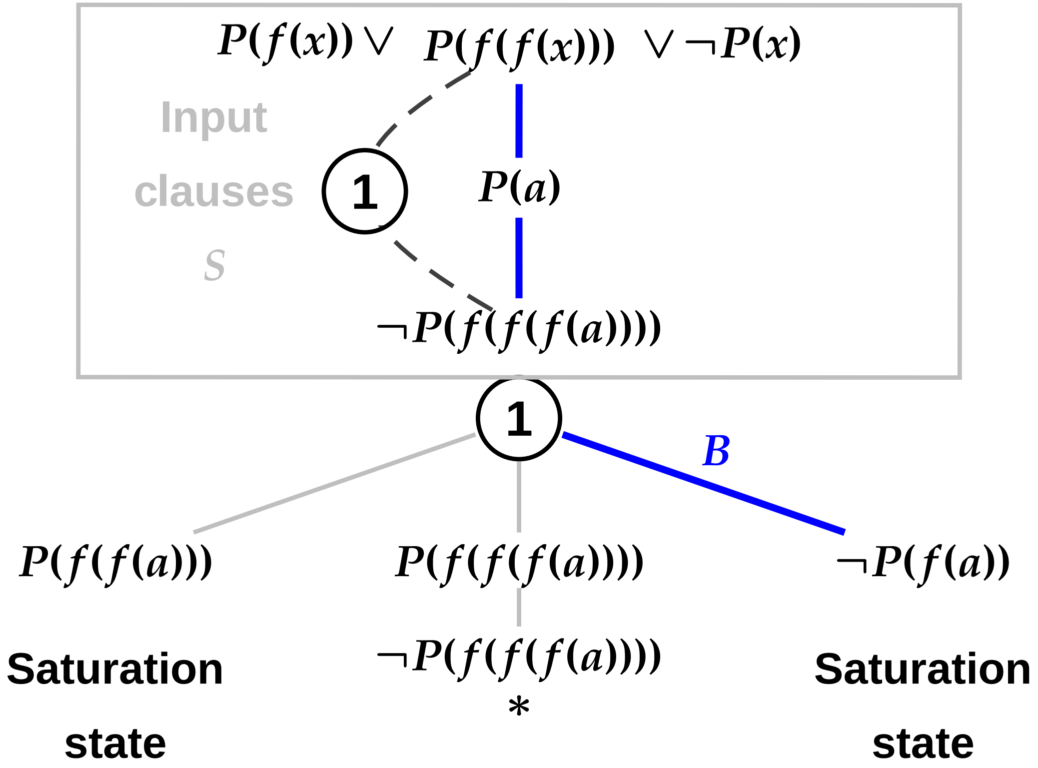
Infinite Herbrand Models

Model extraction also works for infinite Herbrand universes

Infinite Herbrand Models

Model extraction also works for infinite Herbrand universes

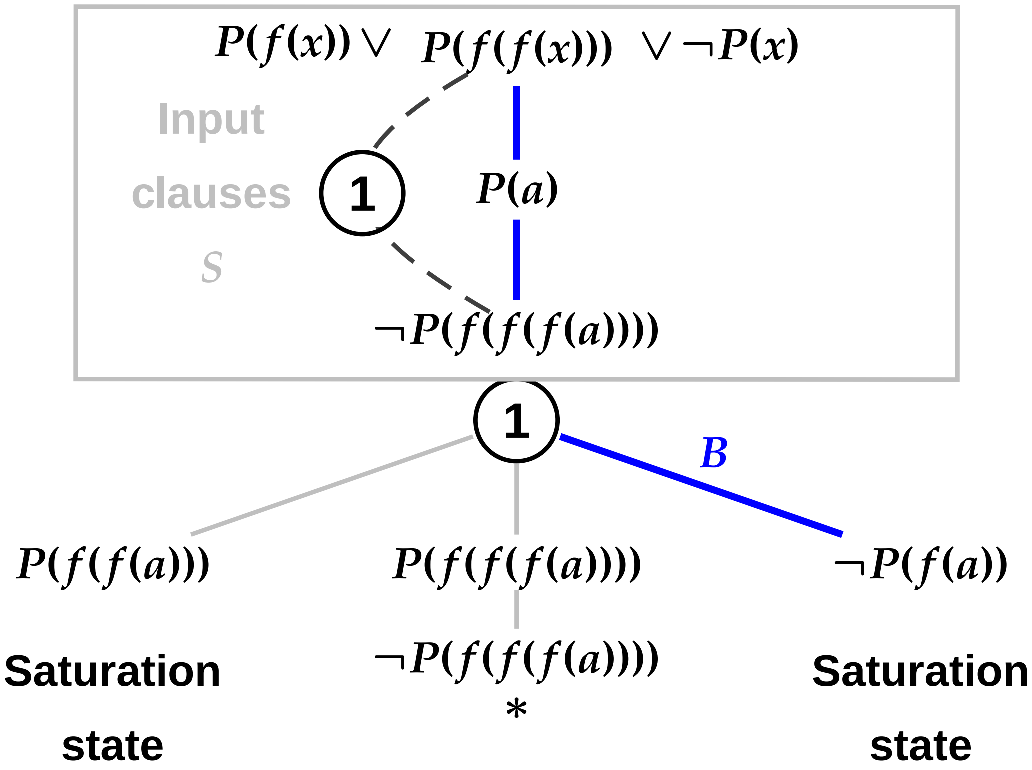
Given a saturated tableau with open branch B:



Infinite Herbrand Models

Model extraction also works for infinite Herbrand universes

Given a saturated tableau with open branch B:



The enumeration for B

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

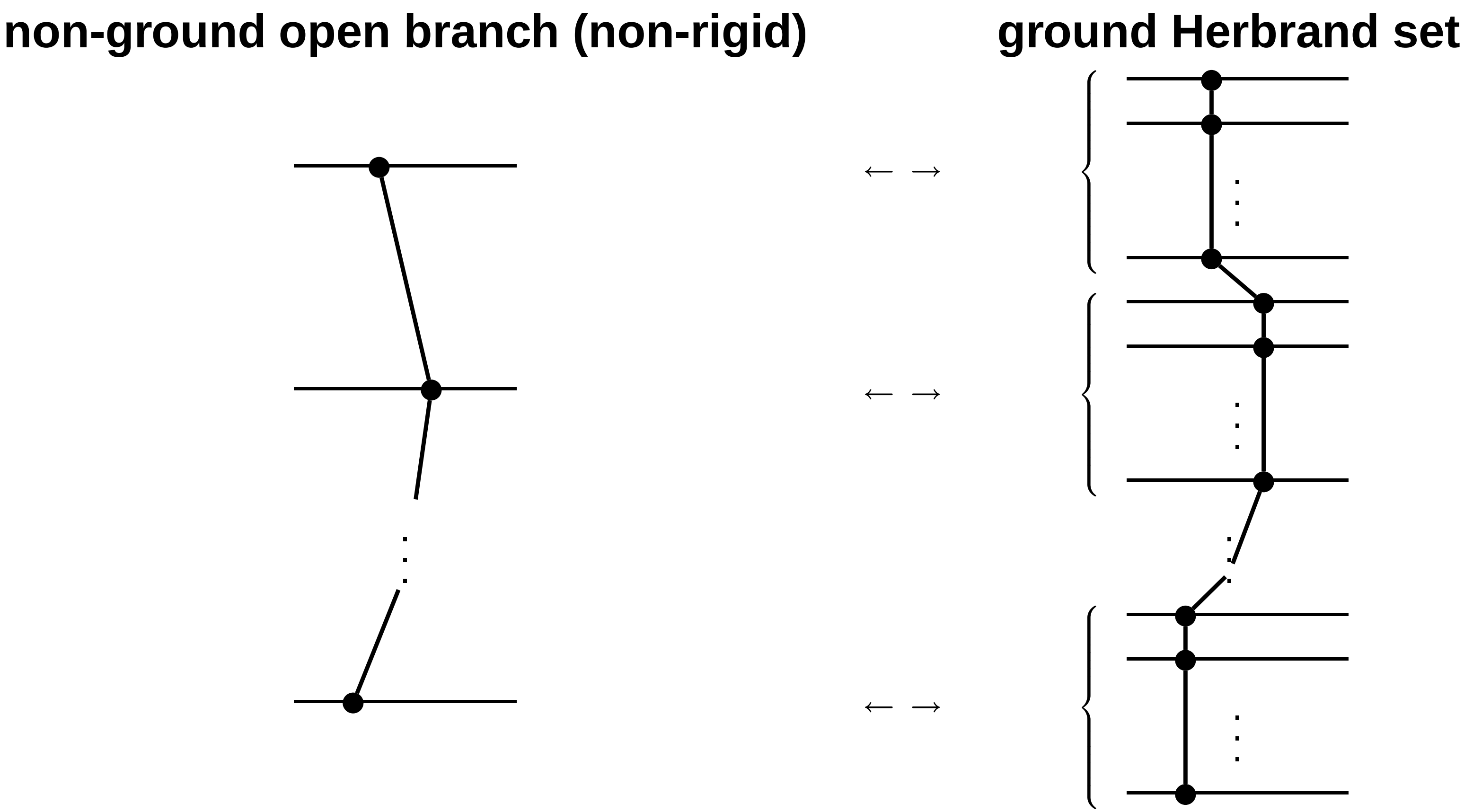
implies a finite representation of an infinite Herbrand model:

$\{\neg P(f(f(f(a)))), \neg P(f(a)), P(a)\}, \{P(f(f(\mathfrak{s})))\}$

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**

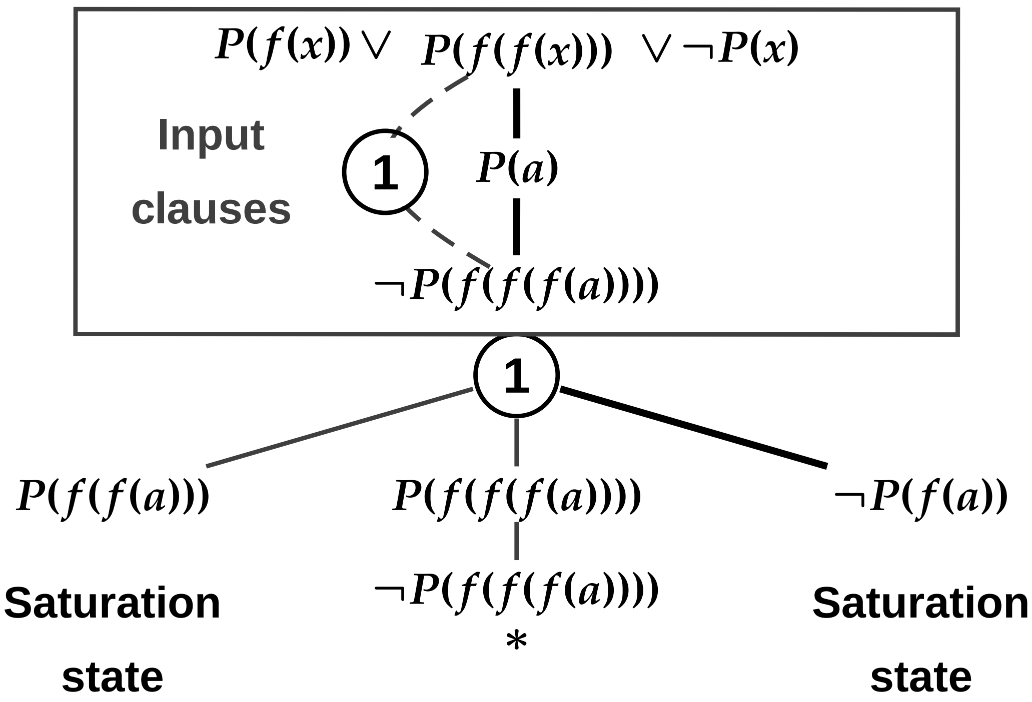


- **Closed ground path corresponds to applicable link**

  $\Leftrightarrow$ contradicts saturation

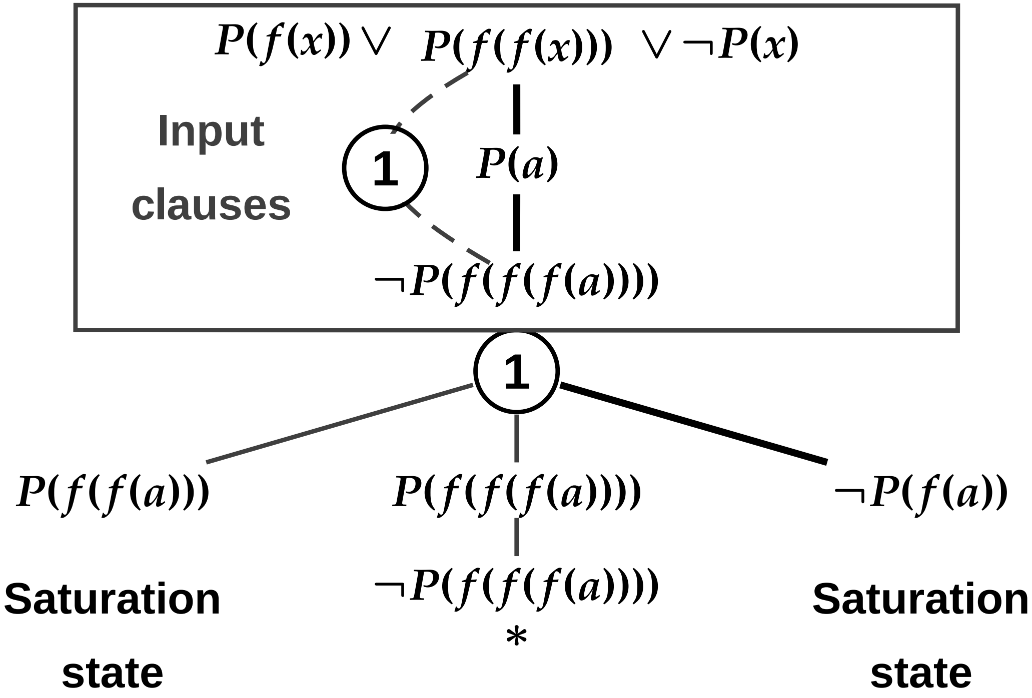
The Saturation Property

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



The Saturation Property

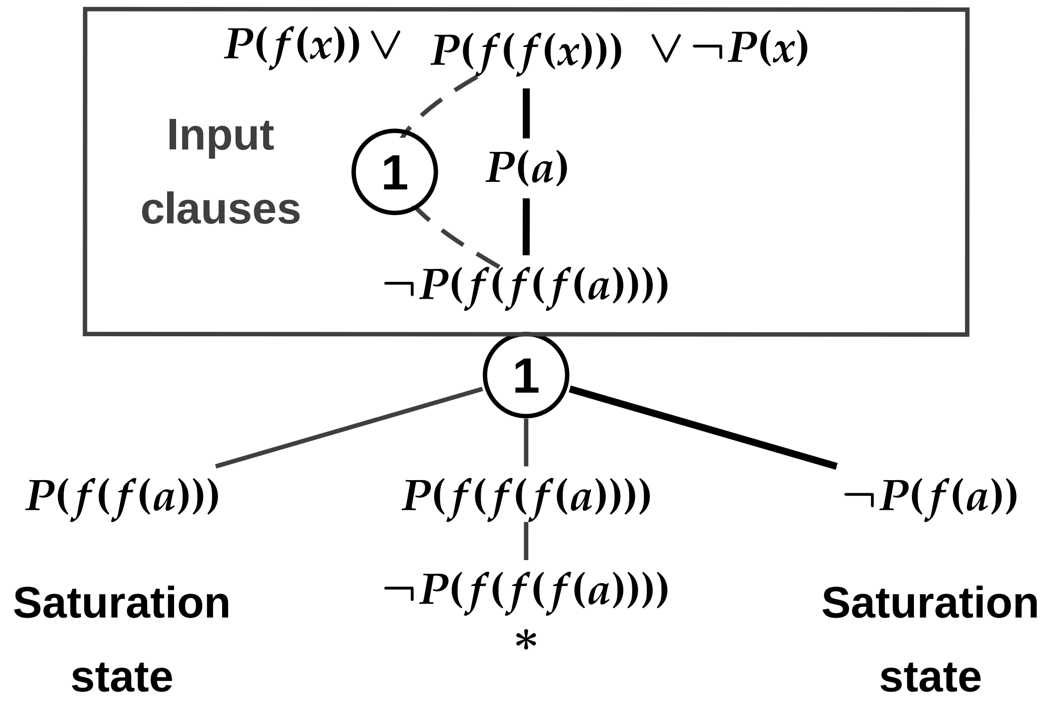
- Saturated open branch specifies a model (only such a branch)
- Model characterised as exception-based representation (EBR)



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

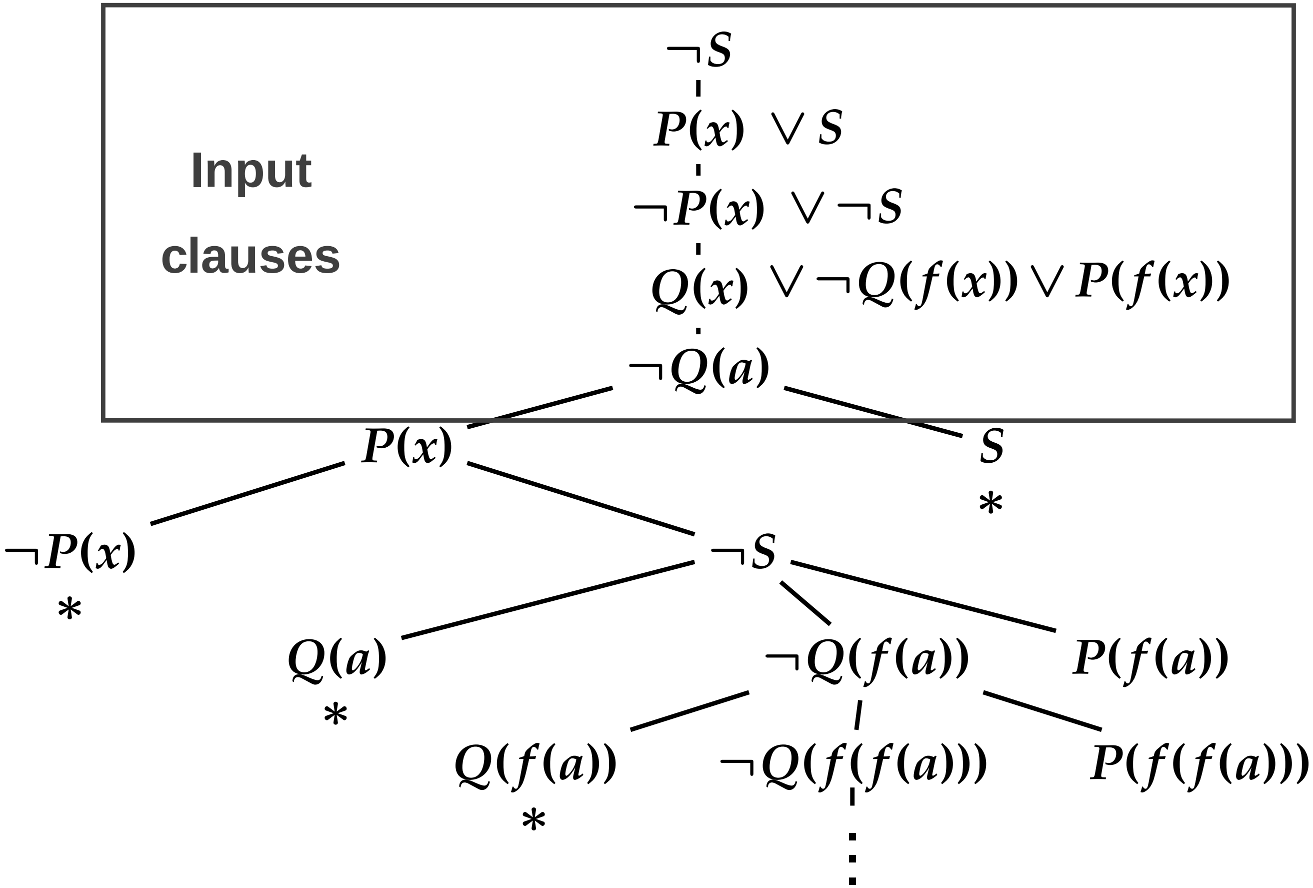
The Saturation Property

- Saturated open branch specifies a model (only such a branch)
- Model characterised as 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 $\sim L$

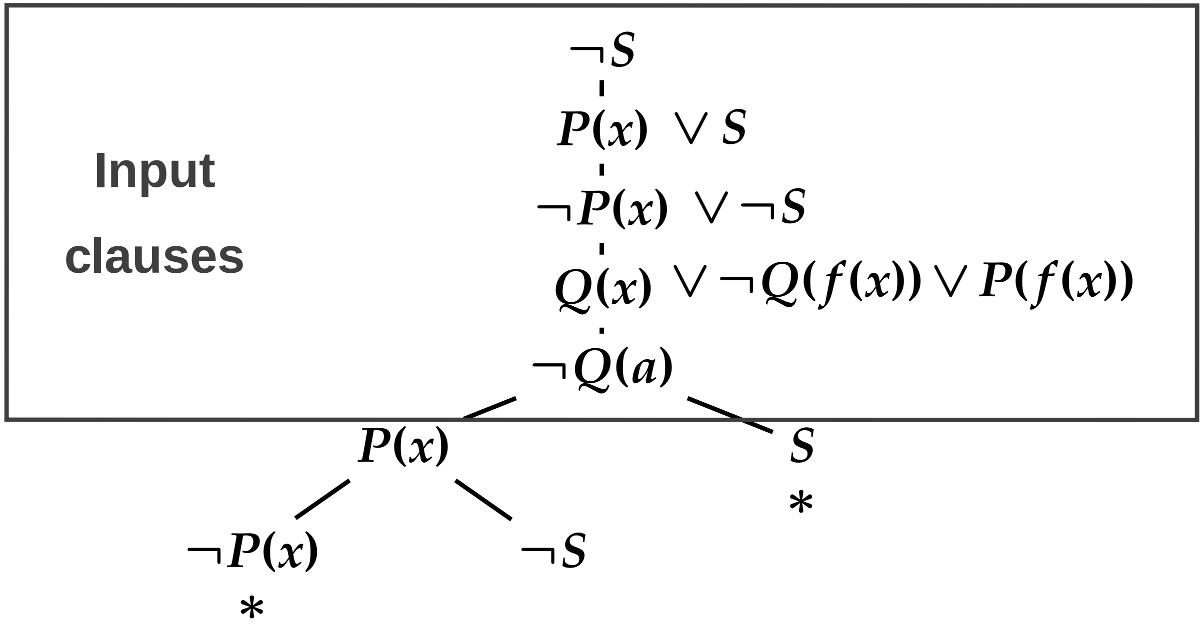
Link Blocking Example

- **The non-termination example revisited**

Input clauses	$\neg S$ $P(x) \vee S$ $\neg P(x) \vee \neg S$ $Q(x) \vee \neg Q(f(x)) \vee P(f(x))$ $\neg Q(a)$

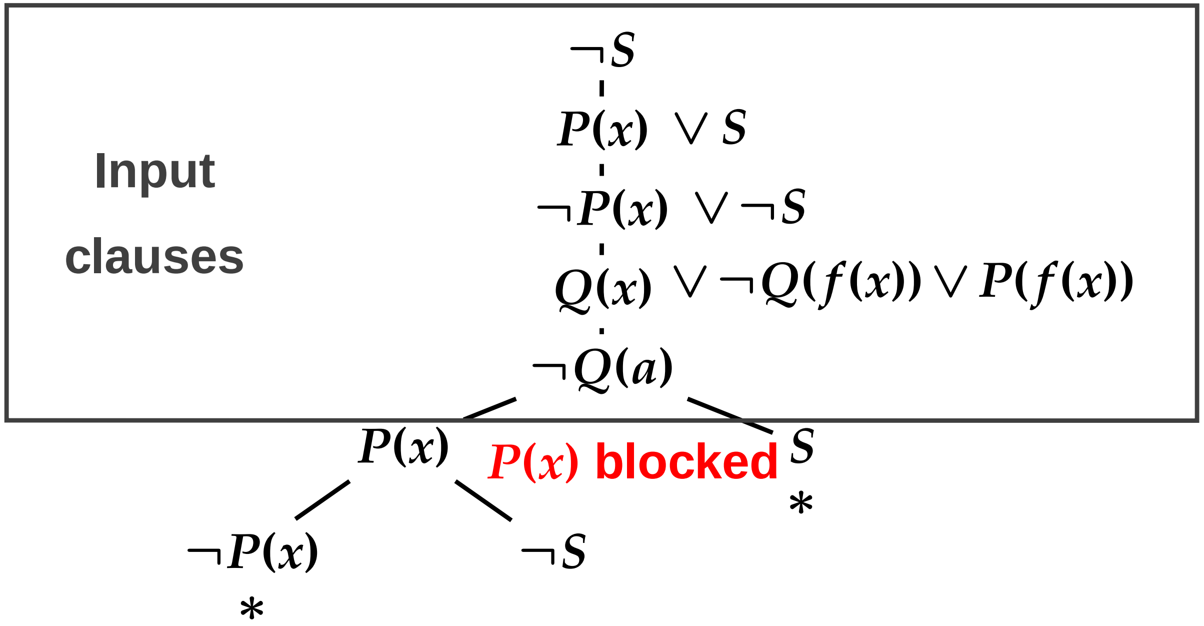
Link Blocking Example

- **The non-termination example revisited**



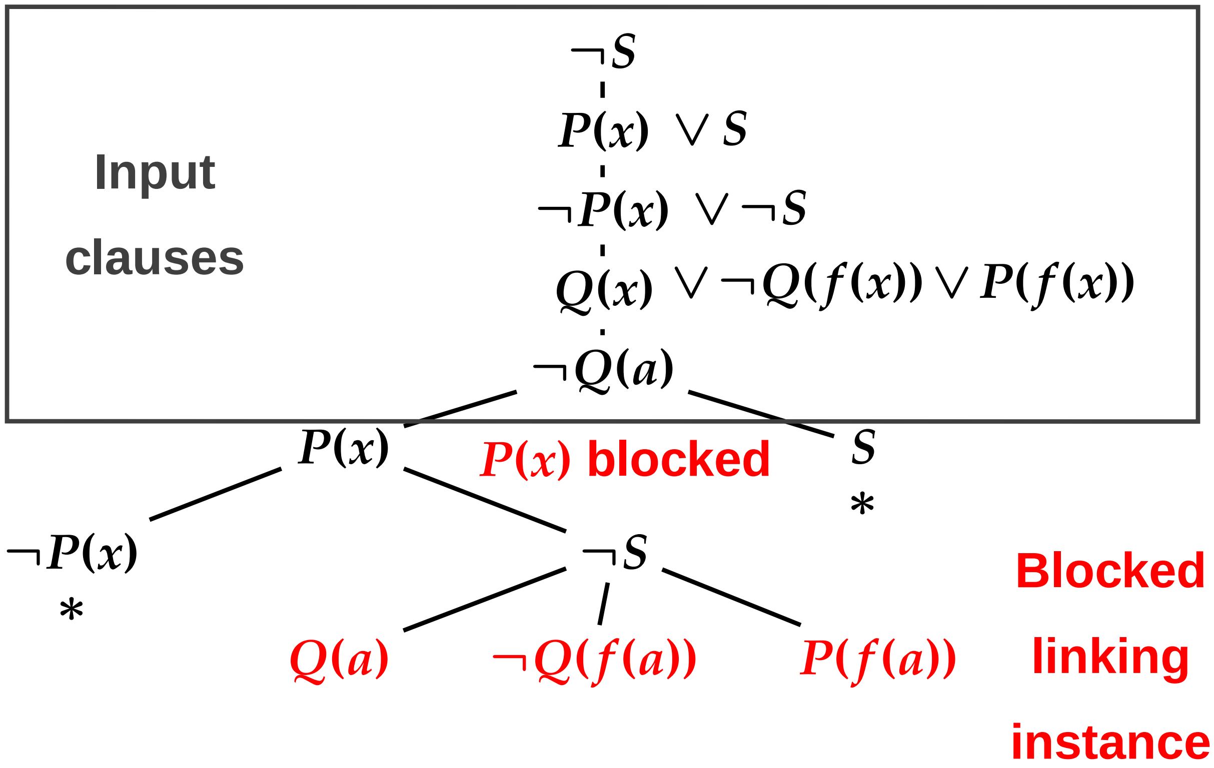
Link Blocking Example

- **The non-termination example revisited**



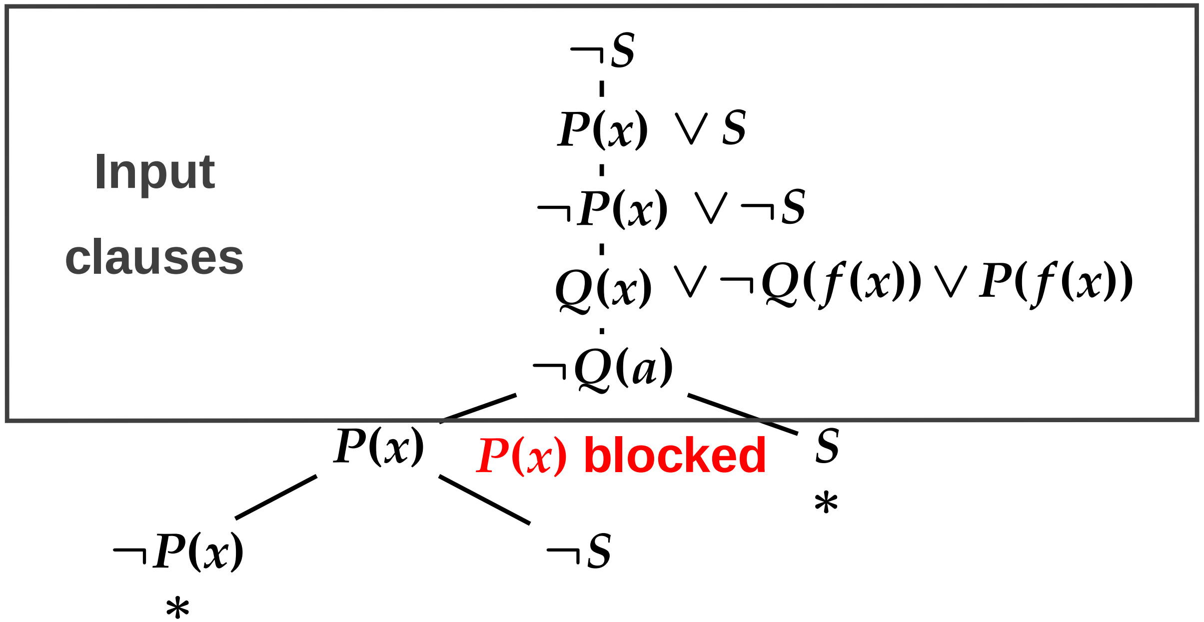
Link Blocking Example

- **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

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

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

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

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

Unsatisfiable clause set

Cyclic Link Blocking

two links

A

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

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

B

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

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

Unsatisfiable clause set

Cyclic Link Blocking

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

blocked **A**

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

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

blocked **B**

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

Unsatisfiable clause set

Cyclic Link Blocking

blocked A $Q(a, y) \vee \neg P(a, y)$

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

blocked B $P(a, y) \vee Q(a, y)$

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

Unsatisfiable clause set

no link applicable

- 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**

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

A

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

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

B

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

Unsatisfiable clause set

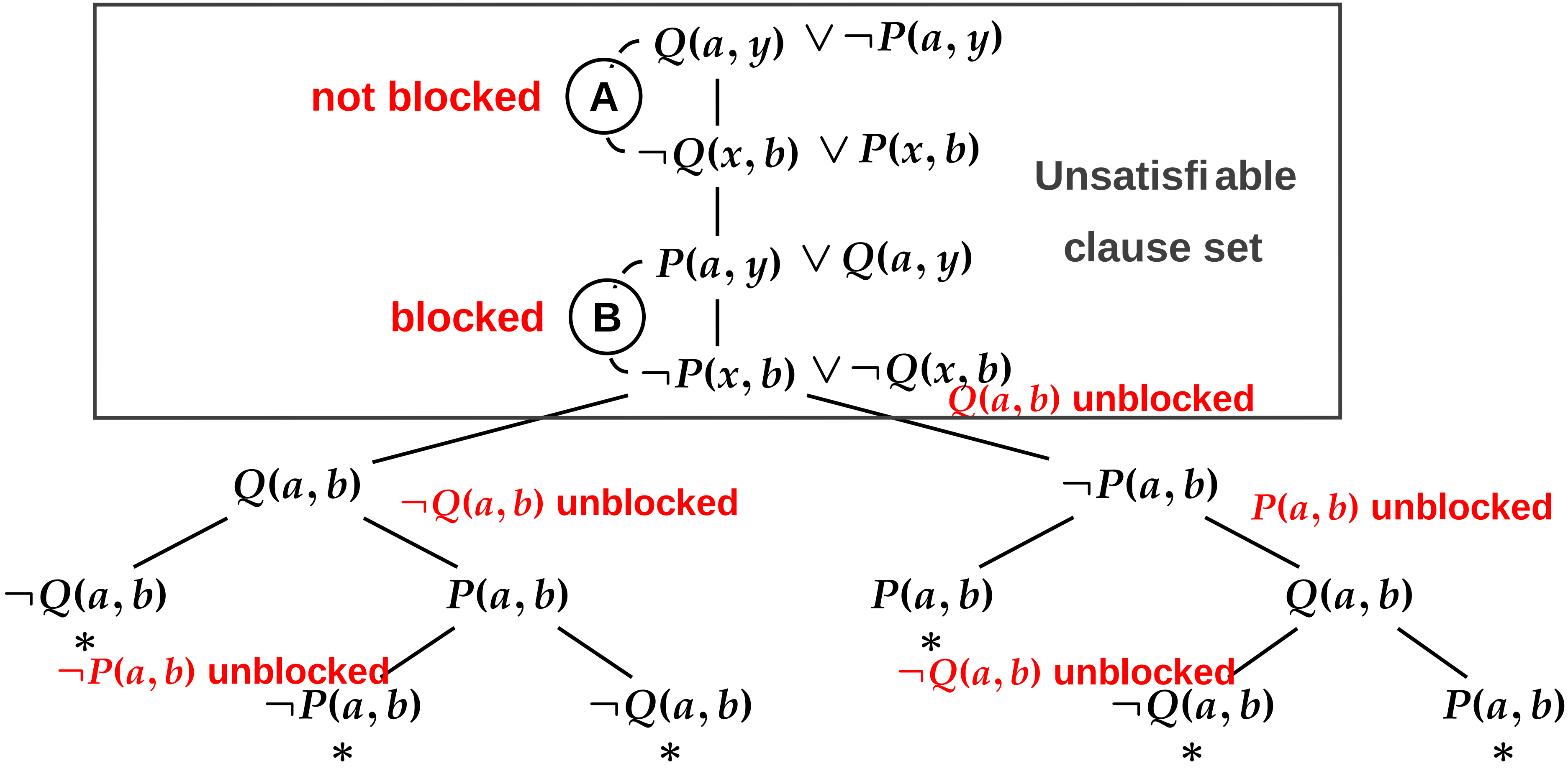
Cyclic Link Blocking Resolved

- **We try again, this time with a blocking ordering**

		$Q(a,y) \vee \neg P(a,y)$	
not blocked	A	|	
		$\neg Q(x,b) \vee P(x,b)$	Unsatisfiable
		|	
		$P(a,y) \vee Q(a,y)$	clause set
blocked	B	|	
		$\neg P(x,b) \vee \neg Q(x,b)$	

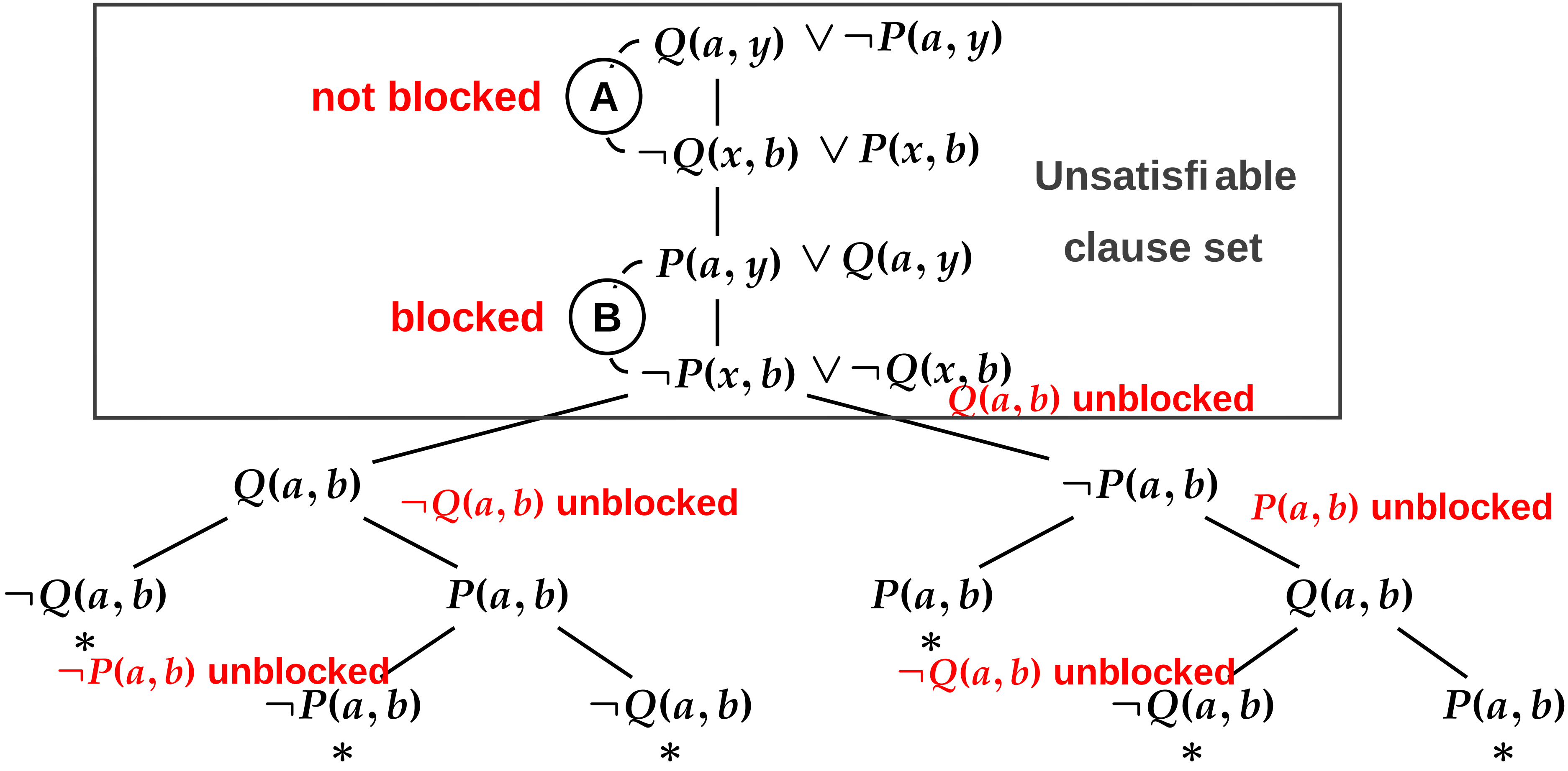
Cyclic Link Blocking Resolved

- 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:

saturated open tableau branch B^+

$$\Longrightarrow$$

consistent path P^* 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 $L\bot$ and $\overline{L}\bot$ (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**

Theory Reasoning and Equality

Theory Reasoning (I)

Problem: Given a theory T and a clause set S. Is S T-unsatisfiable?

Verification applications: T is usually a combination of theories

(arithmetic, arrays, records, ...)

Example: Precondition: $x > 0$

Program: $y := x + 1$

Postcondition: $y > 1$

T is linear integer arithmetic. Show T-validity of

$$\forall x, y\ ((x > 0) \wedge (y = x + 1) \longrightarrow (y > 0))$$

More generally, have to show T-validity of a formula $\forall x\ \phi(x)$

Theory Reasoning (II)

Popular approch to prove T-validity of $\forall x\ \phi(x)$

- **Treat $\phi(x)$ as propositional formula**
- **Use DPLL (BDD, Tableaux, . . .) to get model $\{L_1, \ldots, L_n\}$ of $\phi(x)$**
- **Verify that $\forall(L_1 \wedge \cdots \wedge L_n)$ is T-valid (i.e. L_i's are interpreted again)**
- **The latter can be done for many useful theories (arrays, restricted arithmetic, integers, lists) and also combinations**
- **Bag of techniques to make this approach efficient**

Theory Reasoning (III)

Notation: $\forall x\ \phi(x)$ **is** T**-valid:** $\models_T \forall x\ \phi(x)$

General problem: show T**-validity under assumptions** Γ**:**

$$\Gamma \models_T \forall x\ \phi(x) \qquad\qquad (\Gamma \text{ could be } \forall x\ \psi(x))$$

Example (T **theory of equality, variables universally quantified):**

$$\{f(h(x)) \approx c,\ h(x) \approx x\} \models_T f(a) \not\approx c$$

Propositional reasoning is not enough:

$$\{f(h(\bot)) \approx c,\ h(\bot) \approx \bot\} \not\models_T f(a) \not\approx c$$

Theory Reasoning (III)

Notation: $\forall x\ \phi(x)$ **is** T**-valid:** $\models_T \forall x\ \phi(x)$

General problem: show T**-validity under assumptions** Γ**:**

$$\Gamma \models_T \forall x\ \phi(x) \qquad\qquad (\Gamma \text{ could be } \forall x\ \psi(x))$$

Example (T **theory of equality, variables universally quantified):**

$$\{f(h(x)) \approx c,\ h(x) \approx x\} \models_T f(a) \not\approx c$$

Propositional reasoning is not enough:

$$\{f(h(\bot)) \approx c,\ h(\bot) \approx \bot\} \not\models_T f(a) \not\approx c$$

How to discover required instances $f(h(a)) \approx c$ **and** $h(a) \approx a$ **?**
Propositional reasoning doesn't provide guidance!

Theory Reasoning (IV)

Dilemma:

- **Could enumerate ground instances or make heuristic choice**

  (current practice in verification tools, e.g. CVC Lite)

  - Inefficient, incomplete

  + Can use existing decision procedures for T

- **Use theory reasoner to compute T-unifiers**

  + Possibly complete and efficient, depending from T

  (see below for Inst-Gen with equality)

  - Does not exploit existing decision procedures for T,

  have to design new theory reasoner

Theory Reasoning (IV)

Dilemma:

- **Could enumerate ground instances or make heuristic choice**
  (current practice in verification tools, e.g. CVC Lite)
  - Inefficient, incomplete
  + Can use existing decision procedures for T
- **Use theory reasoner to compute T-unifiers**
  + Possibly complete and efficient, depending from T
  (see below for Inst-Gen with equality)
  - Does not exploit existing decision procedures for T,
  have to design new theory reasoner

Perhaps the most pressing research problem!

Theory Reasoning for Equality

- Equality is by far the most important and mostly used theory
- Unlike other theories handled on the first-order level
- Different ways of integrating equality into instance based methods

Theory Reasoning for Equality

- Equality is by far the most important and mostly used theory
- Unlike other theories handled on the first-order level
- Different ways of integrating equality into instance based methods
- The easiest form: axiomatic equality handling

Theory Reasoning for Equality

- Equality is by far the most important and mostly used theory
- Unlike other theories handled on the first-order level
- Different ways of integrating equality into instance based methods
- The easiest form: axiomatic equality handling
- Other methods all based on paramodulation:

Theory Reasoning for Equality

- Equality is by far the most important and mostly used theory
- Unlike other theories handled on the first-order level
- Different ways of integrating equality into instance based methods
- The easiest form: axiomatic equality handling
- Other methods all based on paramodulation:
  - Superposition-like [Bachmair and Ganzinger, 1994] eq-linking (disconnection calculus)

Theory Reasoning for Equality

- Equality is by far the most important and mostly used theory
- Unlike other theories handled on the first-order level
- Different ways of integrating equality into instance based methods
- The easiest form: axiomatic equality handling
- Other methods all based on paramodulation:
  - Superposition-like [Bachmair and Ganzinger, 1994] eq-linking (disconnection calculus)

Theory Reasoning for Equality

- **Equality is by far the most important and mostly used theory**
- **Unlike other theories handled on the first-order level**
- **Different ways of integrating equality into instance based methods**
- **The easiest form: axiomatic equality handling**
- **Other methods all based on paramodulation:**
  - **Superposition-like [Bachmair and Ganzinger, 1994] eq-linking (disconnection calculus)**
  - **Disagreement linking (disconnection calculus)**
  - **Unit paramodulation and non-proper demodulation (Inst-Gen)**

Axiomatic Equality Handling

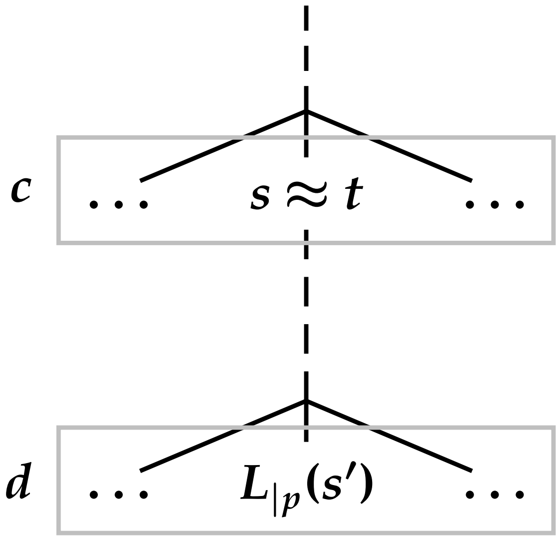
- Simplest form of treating equational problems
- No special inference rules or adaption of calculus/prover required
- Equality axioms added to input clause set
- Axioms for reflexivity, transitivity and symmetry
- Substitution axioms for all functors and predicate symbols. For example:

  $$x \approx y \rightarrow f(\ldots, x, \ldots) \approx f(\ldots, y, \ldots)$$

  for every argument position of every functor f
- Inefficient due to redundancy and incompatibility with orderings
- “Disconnects” altered terms from their clauses

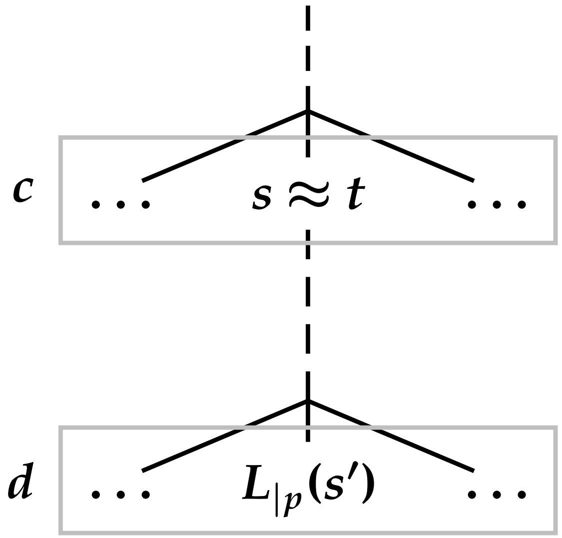
Eq-Linking

- **Additional inference rule: tableau equivalent of paramodulation**



Eq-Linking

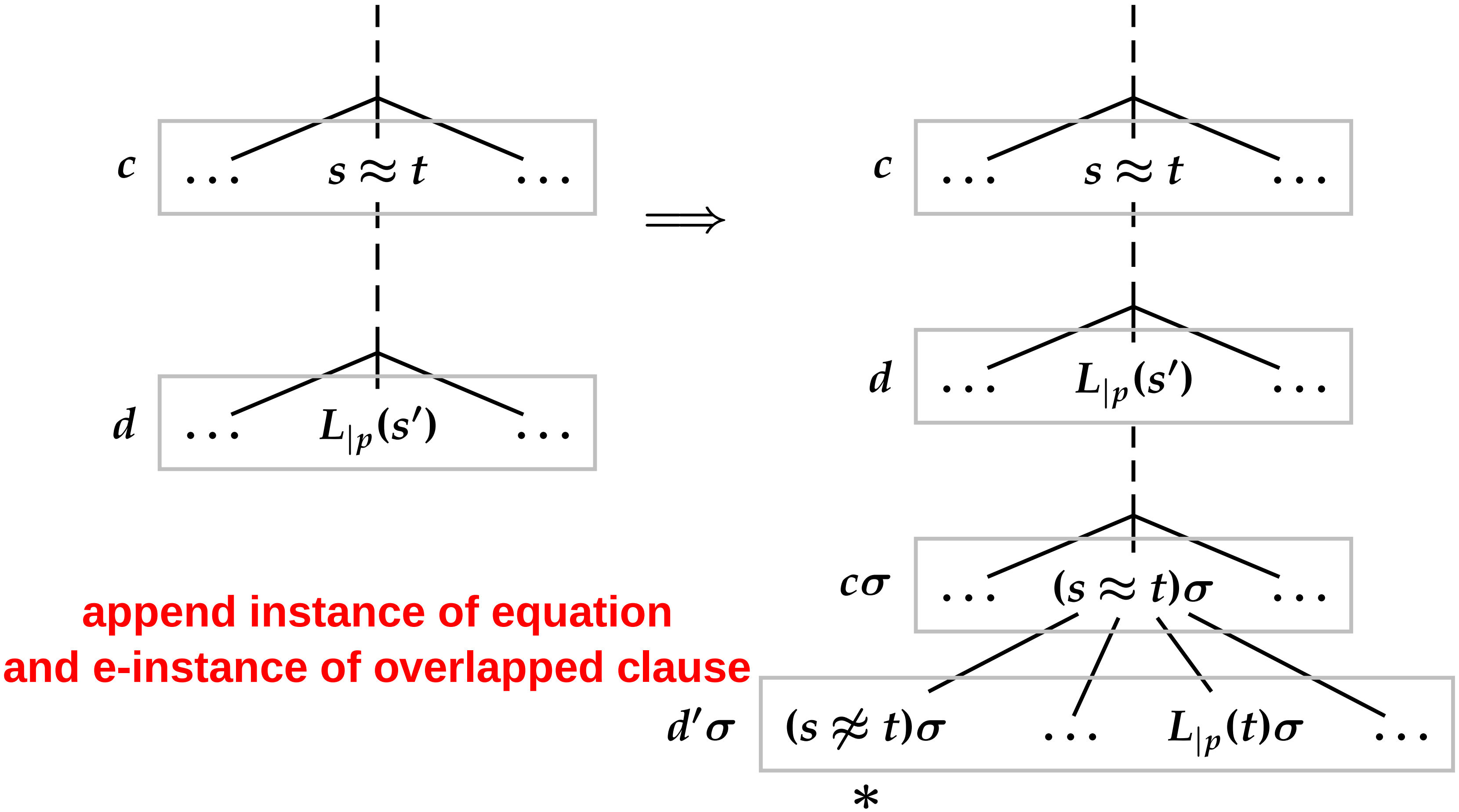
- **Additional inference rule: tableau equivalent of paramodulation**



eq-link on path: one side s of equation and subterm s' unifiable with unifier σ

Eq-Linking

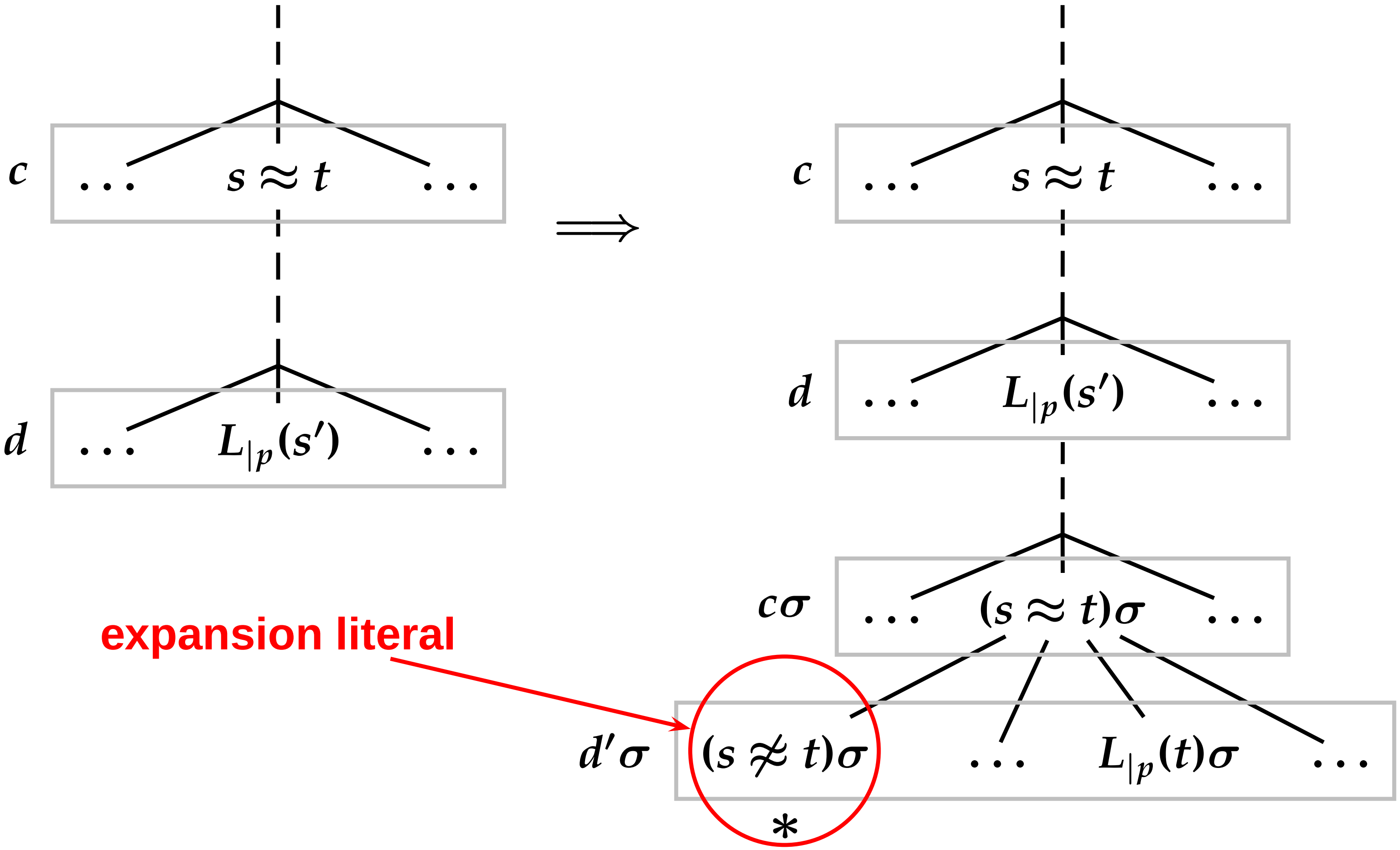
- **Additional inference rule: tableau equivalent of paramodulation**



- **Negation of applied equation added to modified clause: *e-instance***

Eq-Linking

- **Additional inference rule: tableau equivalent of paramodulation**



- **Negation of applied equation added to modified clause: *e-instance***

Eq-Linking (II)

- Overlapping equation and overlapped literal form *eq-link*
- Expansion literals not necessary when eq-linking with unit equations
- Reflexivity linking rule required for completeness:

$$\frac{C \vee s \not\approx t}{C\sigma} \text{ where } \sigma \text{ is the most general unifier of } s \text{ and } t$$

- Unrestricted application of eq-linking introduces large amount of redundancy
- But: eq-linking also compatible with *term orderings*
- Ordered eq-linking allows destructive rewriting of subgoals

An Example Proof with Eq-Linking

Input

clauses

$$h(a) \not\approx c$$
$$h(e) \approx d$$
$$h(f) \approx h(b)$$
$$a \approx b$$
$$c \approx d$$
$$e \approx f$$

An Example Proof with Eq-Linking

Input

clauses

$$h(a) \not\approx c$$

$$h(e) \approx d$$

$$h(f) \approx h(b)$$

$$a \approx b$$

$$c \approx d$$

$$e \approx f$$

$$h(b) \not\approx c$$

An Example Proof with Eq-Linking

Input clauses

$$h(a) \not\approx c$$

$$h(e) \approx d$$

$$h(f) \approx h(b)$$

$$a \approx b$$

$$c \approx d$$

$$e \approx f$$

$$h(b) \not\approx c$$

$$h(f) \approx d$$

An Example Proof with Eq-Linking

Input clauses

$$h(a) \not\approx c$$

$$h(e) \approx d$$

$$h(f) \approx h(b)$$

$$a \approx b$$

$$c \approx d$$

$$e \approx f$$

$$h(b) \not\approx c$$

$$h(f) \approx d$$

$$h(f) \not\approx c$$

An Example Proof with Eq-Linking

Input clauses

$$h(a) \not\approx c$$
$$|$$
$$h(e) \approx d$$
$$|$$
$$h(f) \approx h(b)$$
$$|$$
$$a \approx b$$
$$|$$
$$c \approx d$$
$$|$$
$$e \approx f$$
$$|$$
$$h(b) \not\approx c$$
$$|$$
$$h(f) \approx d$$
$$|$$
$$h(f) \not\approx c$$
$$|$$
$$c \not\approx d$$
$$*$$

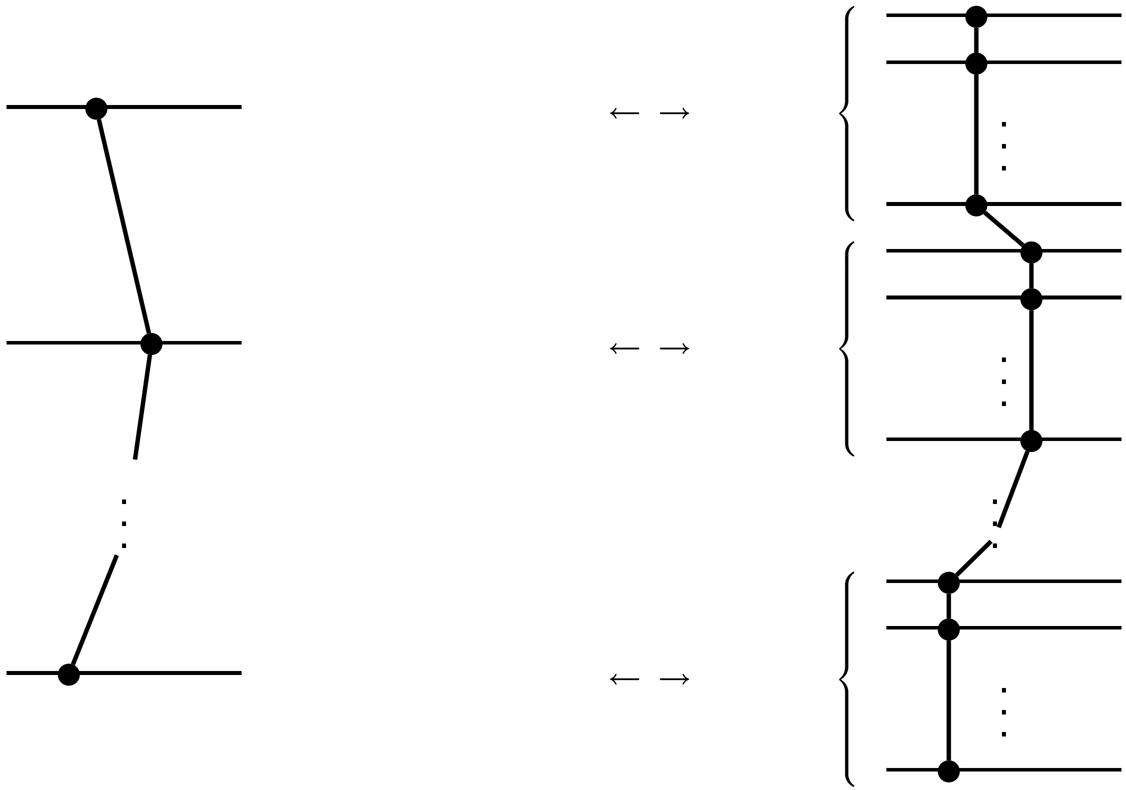
Completeness and Equality (I)

- **Basic concept: open saturated branch represents partial model**

Completeness and Equality (I)

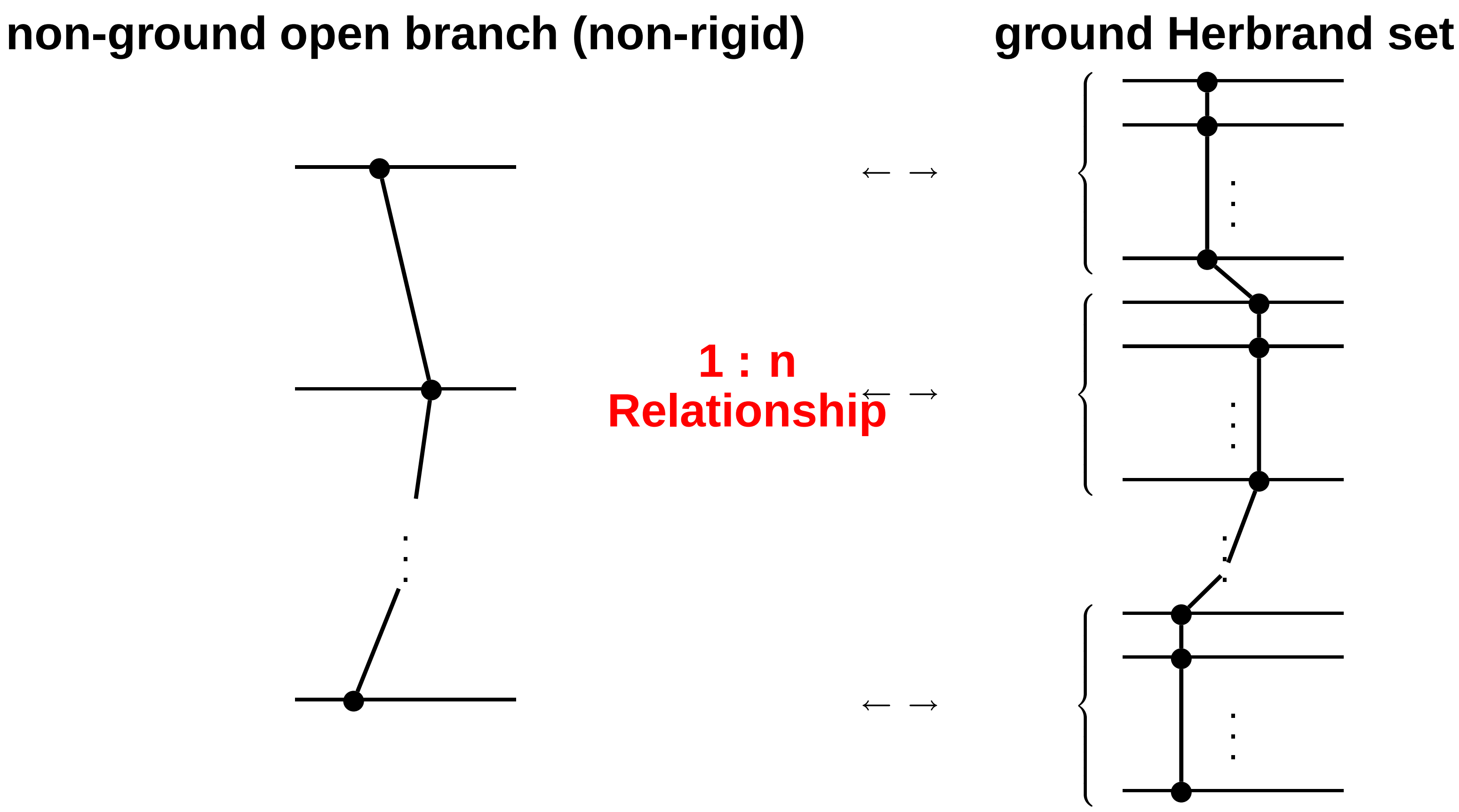
- **Basic concept: open saturated branch represents partial model**
- **Non-equational case: branch determines path through Herbrand set**

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



Completeness and Equality (I)

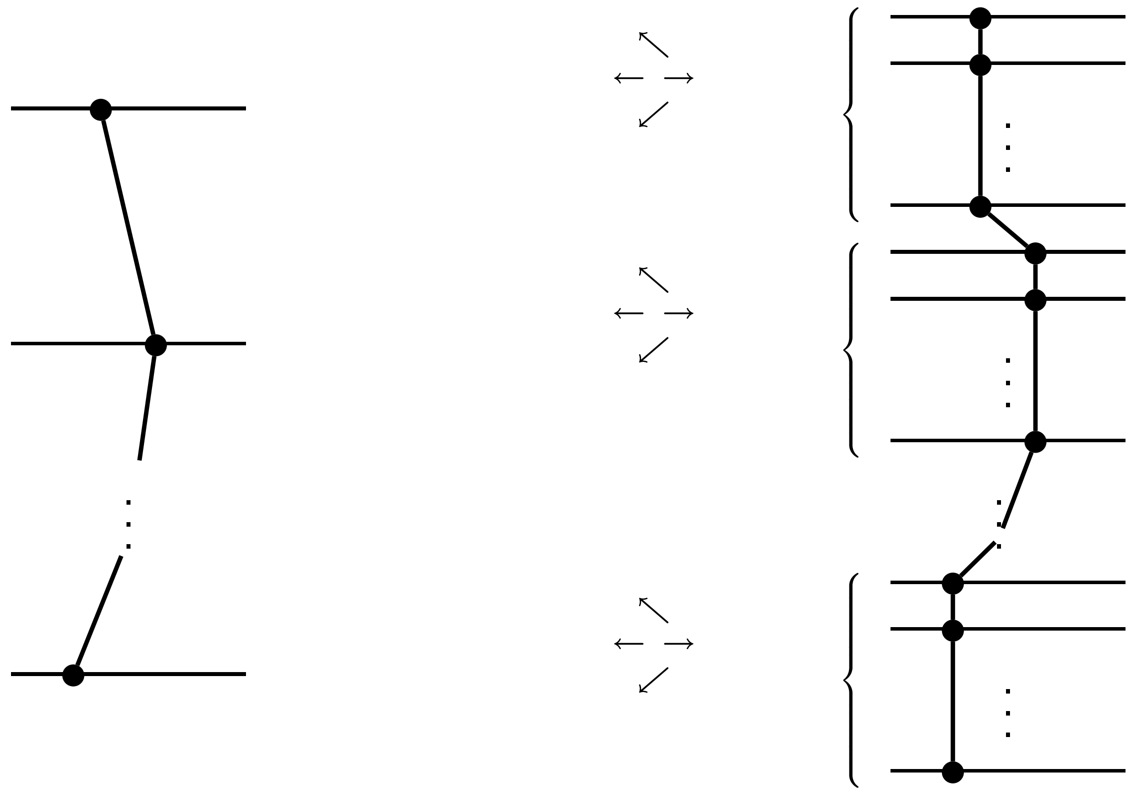
- **Basic concept: open saturated branch represents partial model**
- **Non-equational case: branch determines path through Herbrand set**



Completeness and Equality (II)

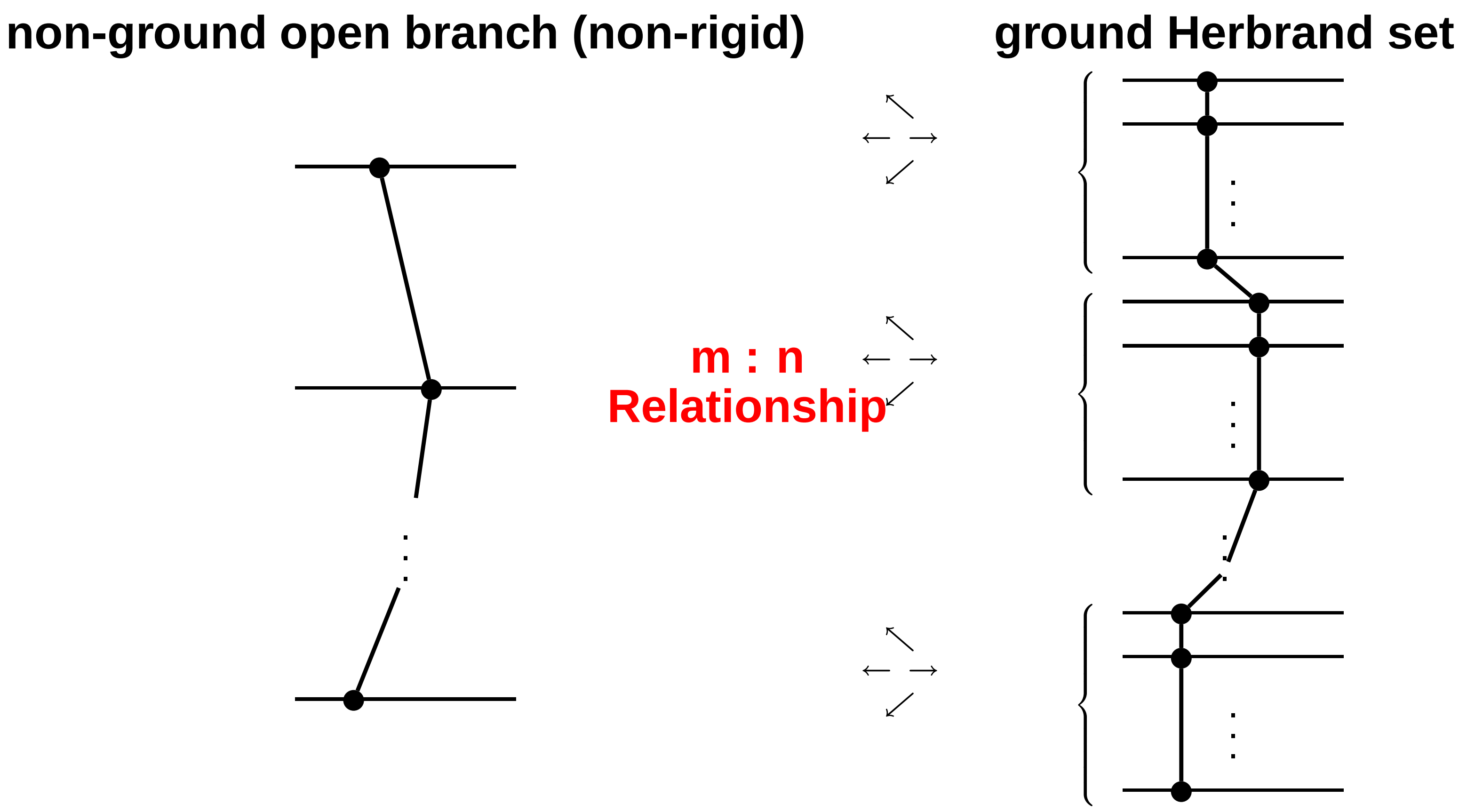
- **Now: one ground clause may correspond to many branch e-variants**

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



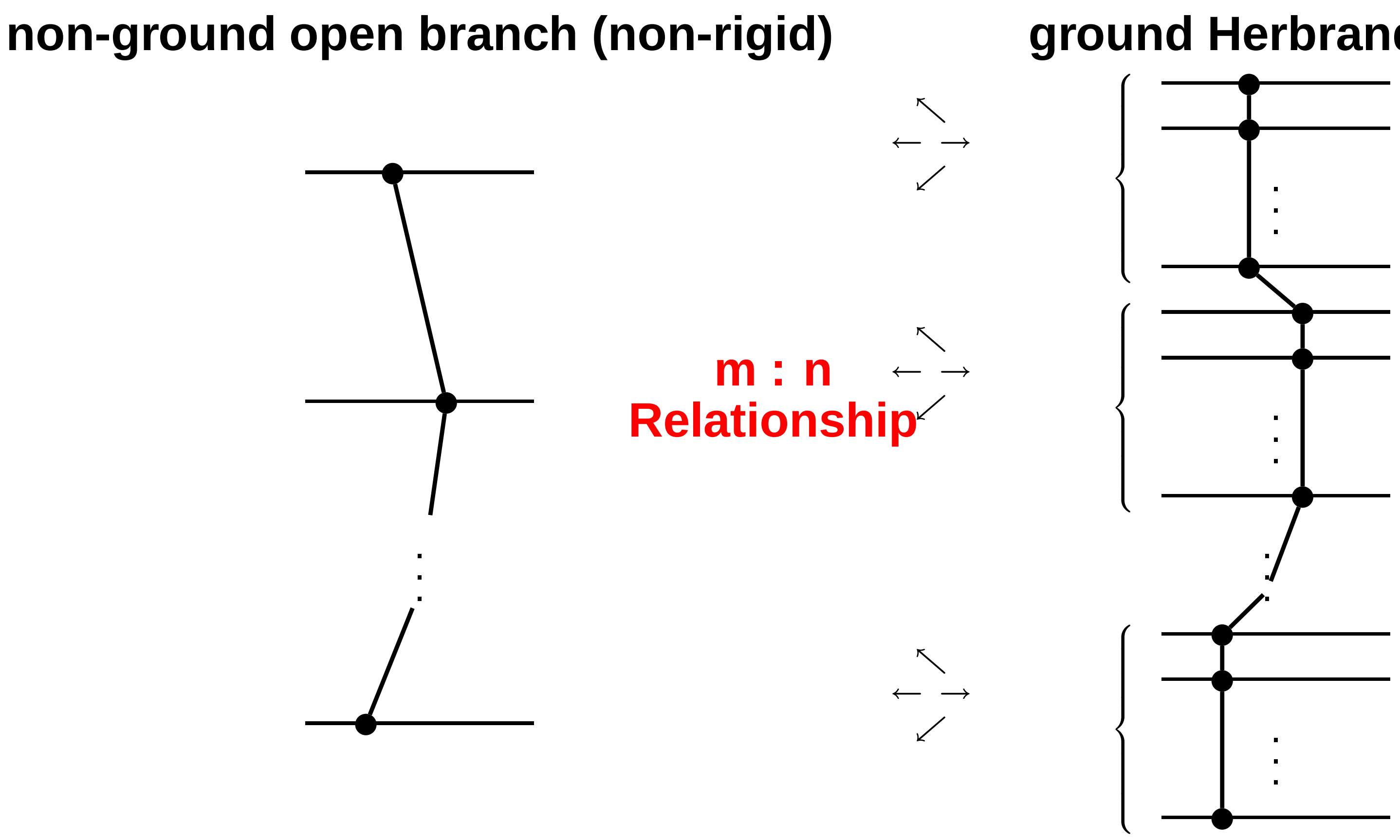
Completeness and Equality (II)

- **Now: one ground clause may correspond to many branch e-variants**



Completeness and Equality (II)

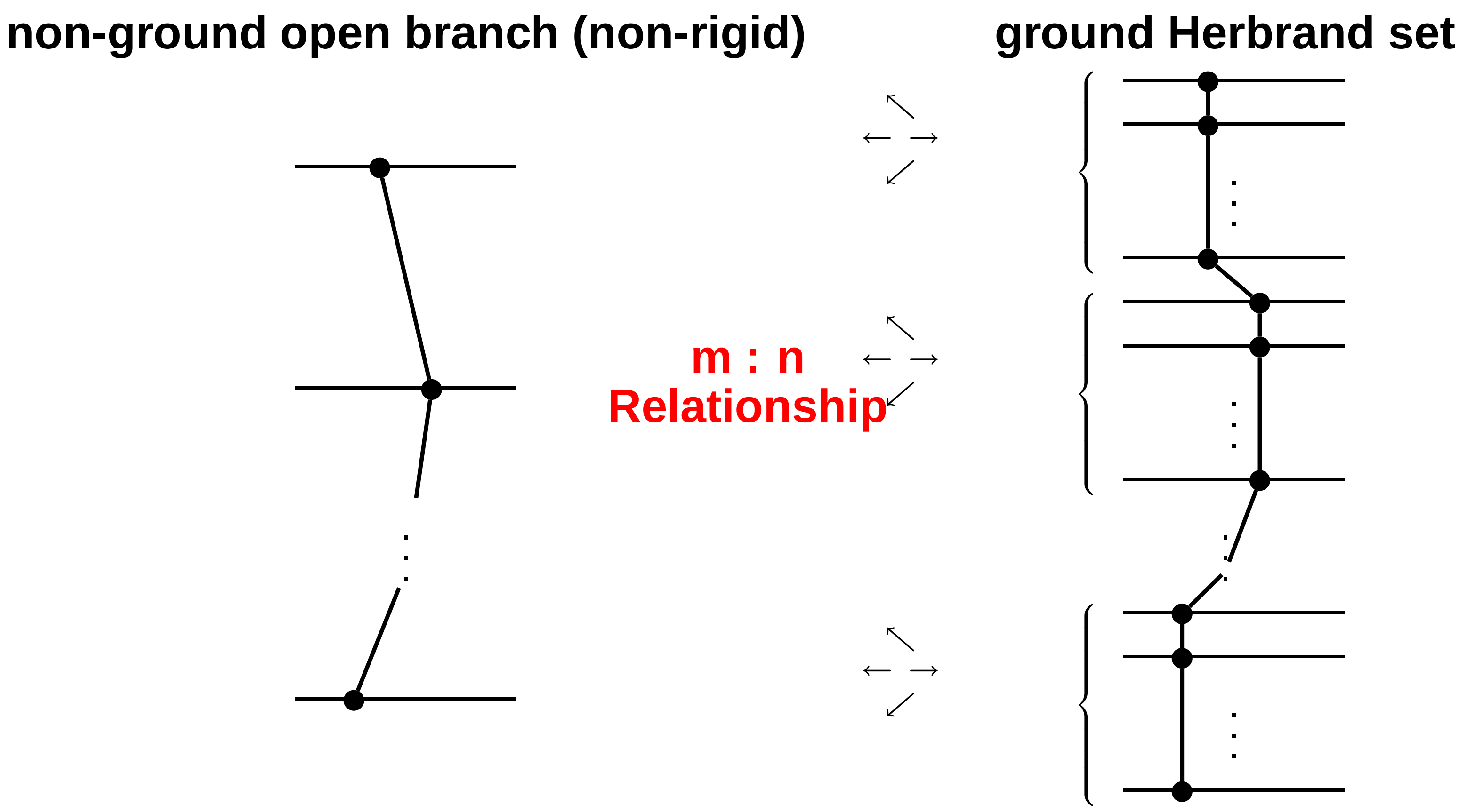
- Now: one ground clause may correspond to many branch e-variants



- Branch may pass through different literals in each of these e-variants

Completeness and Equality (II)

- Now: one ground clause may correspond to many branch e-variants



- Branch may pass through different literals in each of these e-variants
- One representative for each set of e-variants needs to be selected

Eager Variable Elimination

- Given: clause c with literal $l = x \not\approx t$ (x does not occur in t)
- l is a condition for the rest of the clause: $x \approx t \rightarrow c \setminus \{l\}$
- *Eager variable elimination* as a deterministic inference rule:

$$\frac{x \not\approx t \vee k_1 \vee \ldots \vee k_n}{k_1 \vee \ldots \vee k_n\{x/t\}}$$

- Helps keeping clause sizes down
- Care must be taken when eq-linking with unit equations
- Preservation of completeness is still an open problem

  [Gallier and Snyder, 1989]

Disagreement Linking

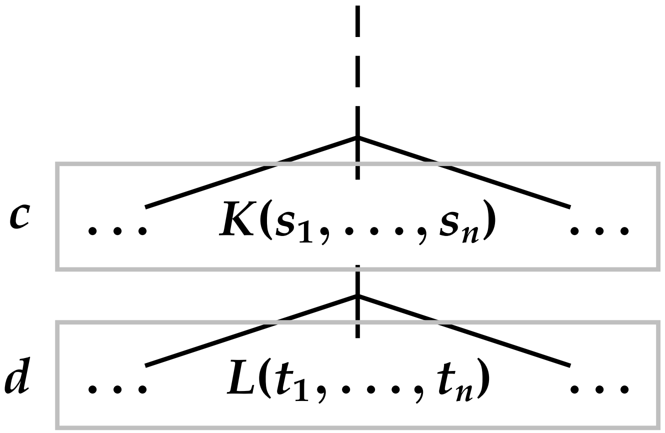
- Inspired by RUE-resolution [Digricoli and Harrison, 1986] and lazy paramodulation [Gallier and Snyder, 1989]
- Similar in behaviour to Brand- and STE-modification on the fly
- Based on the concept of *disagreement sets*:

  $L(s_1, \ldots, s_n)$ and $L(t_1, \ldots, t_n)$, $n \geq 0$ terms or literals

  Disagreement set: $\{s_1 \not\approx t_1, \ldots, s_n \not\approx t_n\}$
- Top-level unification of variable terms: disagreement substitution
- Eager variable elimination performed on disagreement set

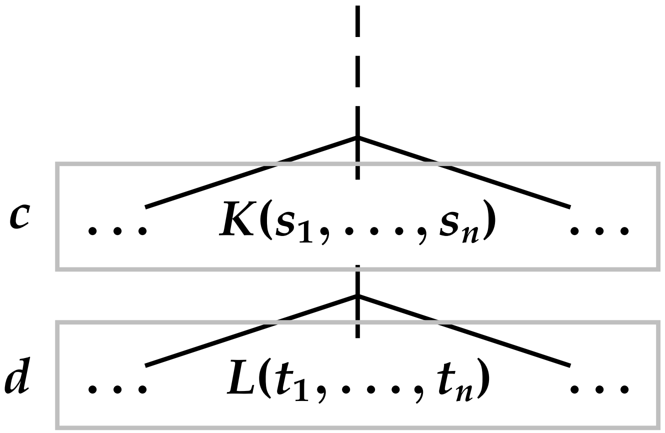
Disagreement Linking (II)

- **Inference rule:**



Disagreement Linking (II)

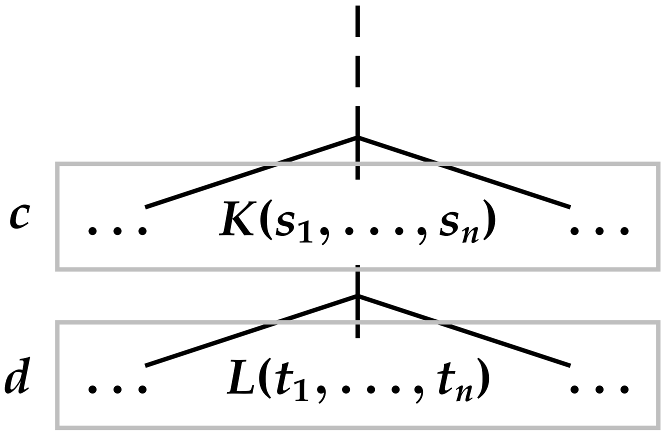
- **Inference rule:**



L and K share the same predicate symbol but have complementary signs

Disagreement Linking (II)

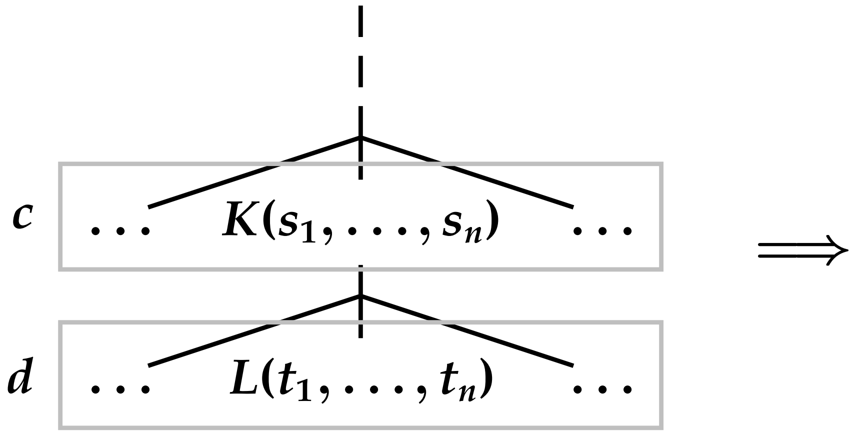
- **Inference rule:**

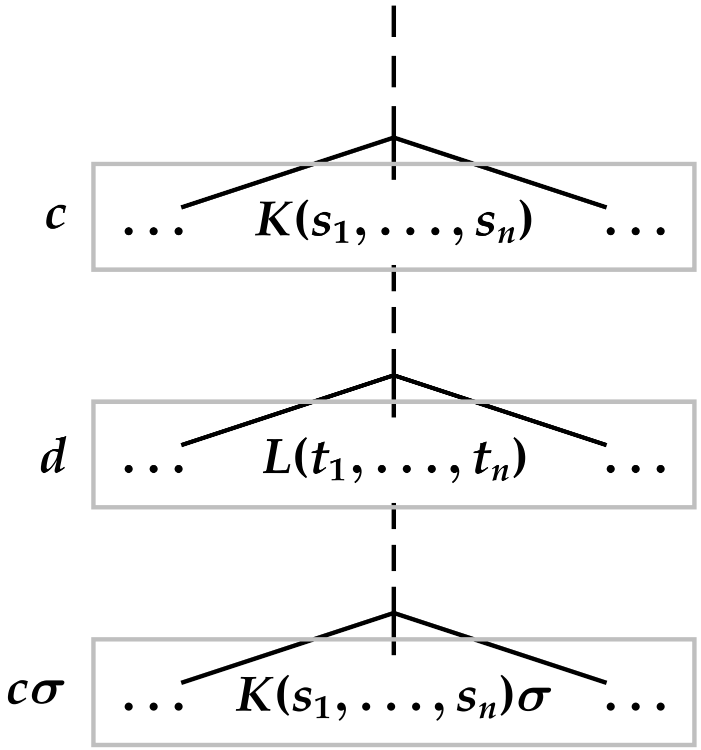


Some of the s_i and t_j are variables
forming the
disagreement substitution σ

Disagreement Linking (II)

- **Inference rule:**

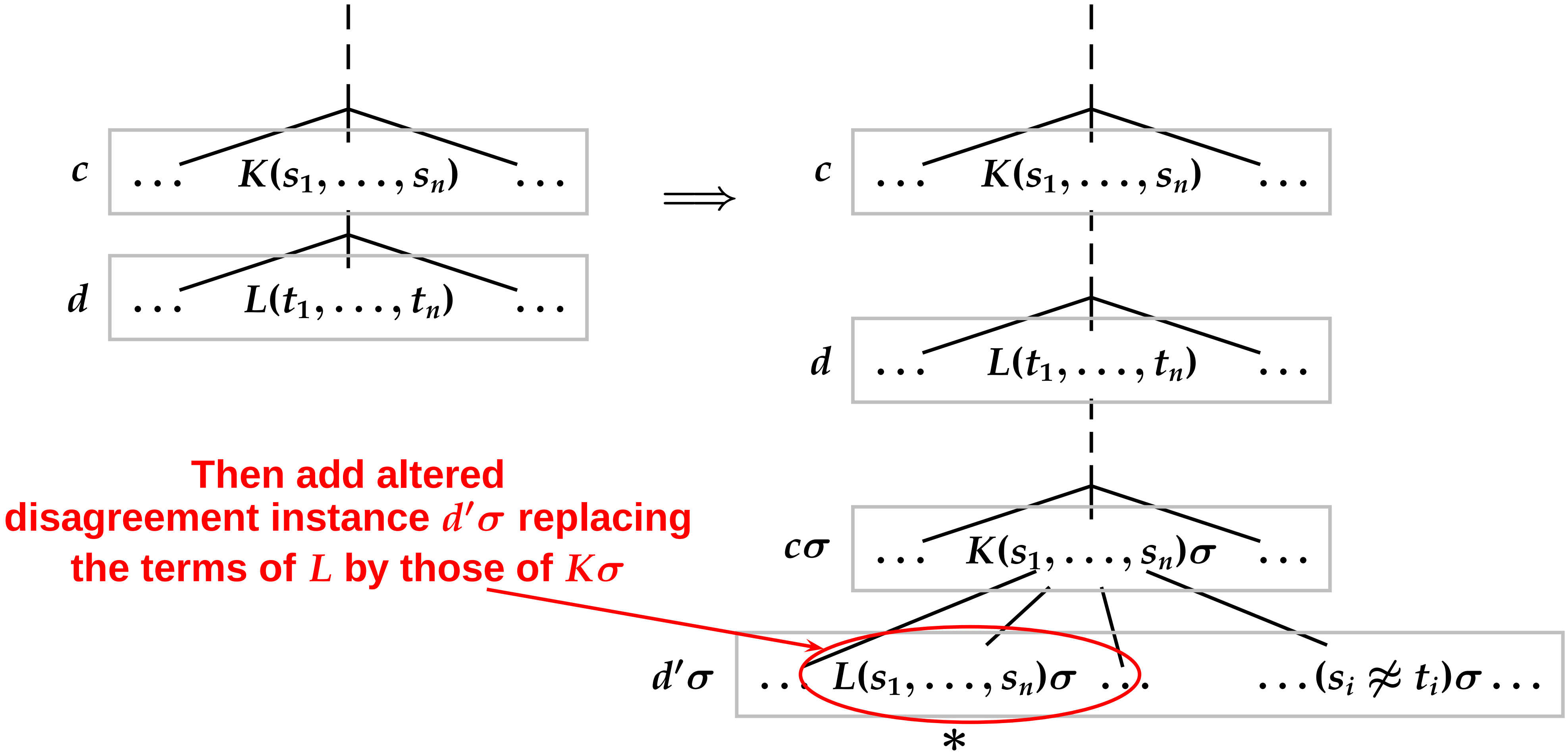




Expand tableau first
by disagreement instance $c\sigma$

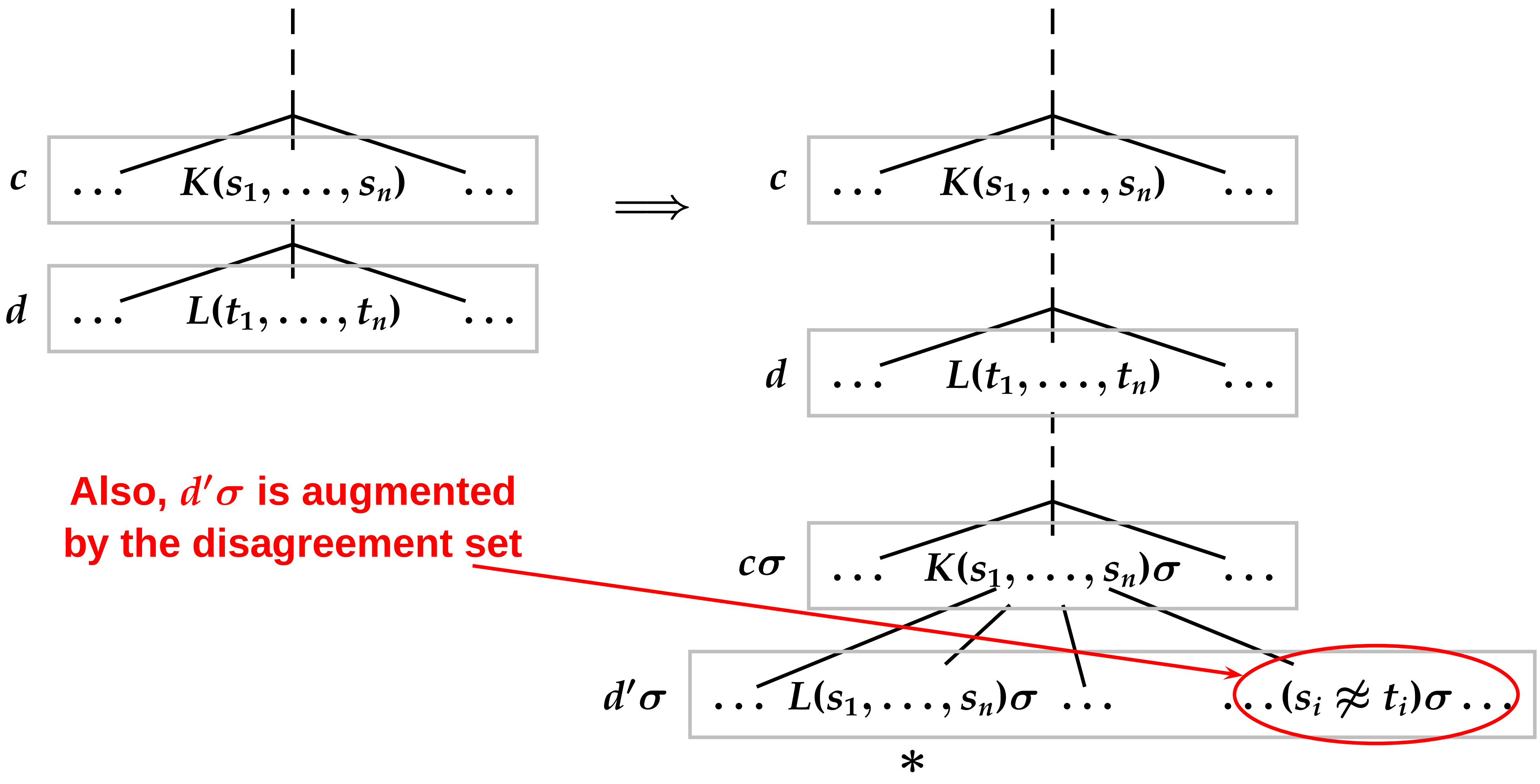
Disagreement Linking (II)

- **Inference rule:**



Disagreement Linking (II)

- **Inference rule:**



Disagreement Linking (III)

- **Decomposition rule required for completeness:**

$$\frac{f(s_1, \ldots, s_n) \not\approx f(t_1, \ldots, t_n)}{s_1 \not\approx t_1 \vee \ldots \vee s_n \not\approx t_n}$$

- **Also, Imitation rule for disequations of the form $x \neq f(x)$**
- **Incompatible with term orderings**
- **Disagreement linking cannot simulate full unification**
- **Additional standard linking necessary for instantiating terms**
- **Explicit symmetry handling required**
- **Sometimes improved recognition of e-satisfiability**

Inst-Gen and Equational Reasoning

- Recently developed equational reasoning for Inst-Gen [Ganzinger and Korovin, 2004]
- New method maintains separation of instance generation and ground satisfiability checking
- Instance generation not by linking, but by paramodulation rules
- Paramodulation performed on selected units
- Sound and complete
- Various techniques of redundancy elimination available

Inst-Gen and Equational Reasoning (II)

f.o. clauses

S

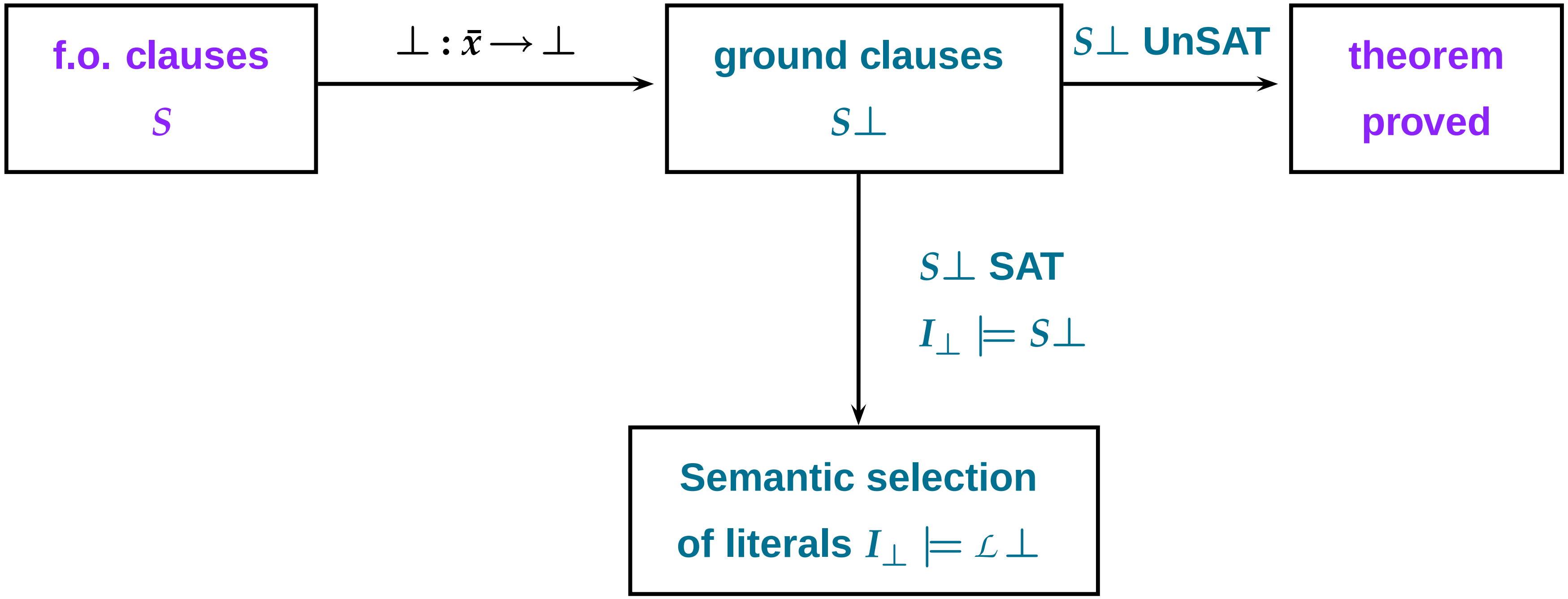
Inst-Gen and Equational Reasoning (II)

$$\begin{array}{ccc} \boxed{\begin{array}{c}\text{f.o. clauses}\\ S\end{array}} & \xrightarrow{\perp : \bar{x} \longrightarrow \perp} & \boxed{\begin{array}{c}\text{ground clauses}\\ S\perp\end{array}} \end{array}$$

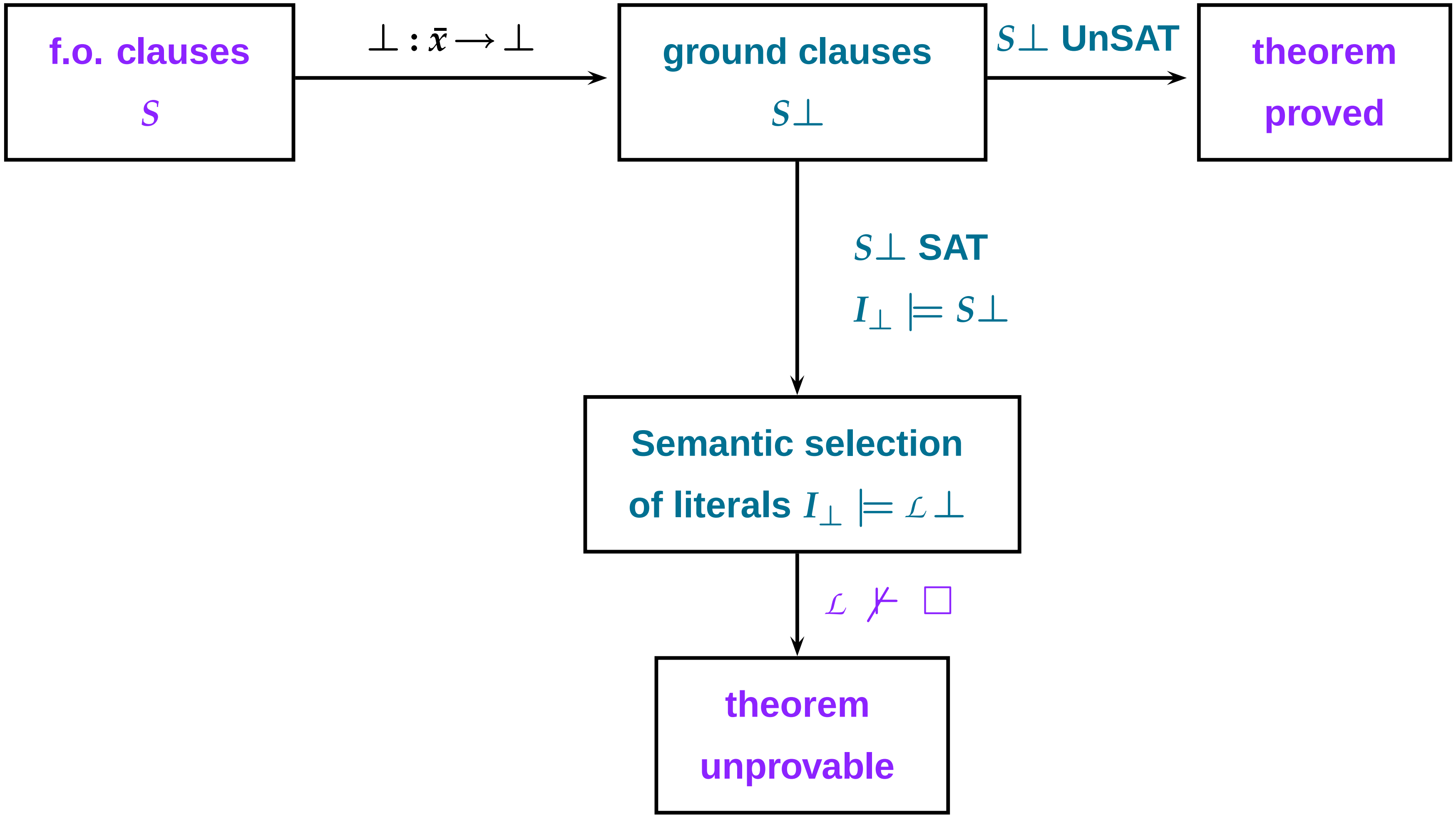
Inst-Gen and Equational Reasoning (II)

$$\boxed{\begin{array}{c}\textbf{f.o. clauses}\\ S\end{array}} \xrightarrow{\perp : \bar{x} \longrightarrow \perp} \boxed{\begin{array}{c}\textbf{ground clauses}\\ S\perp\end{array}} \xrightarrow{S\perp \textbf{ UnSAT}} \boxed{\begin{array}{c}\textbf{theorem}\\ \textbf{proved}\end{array}}$$

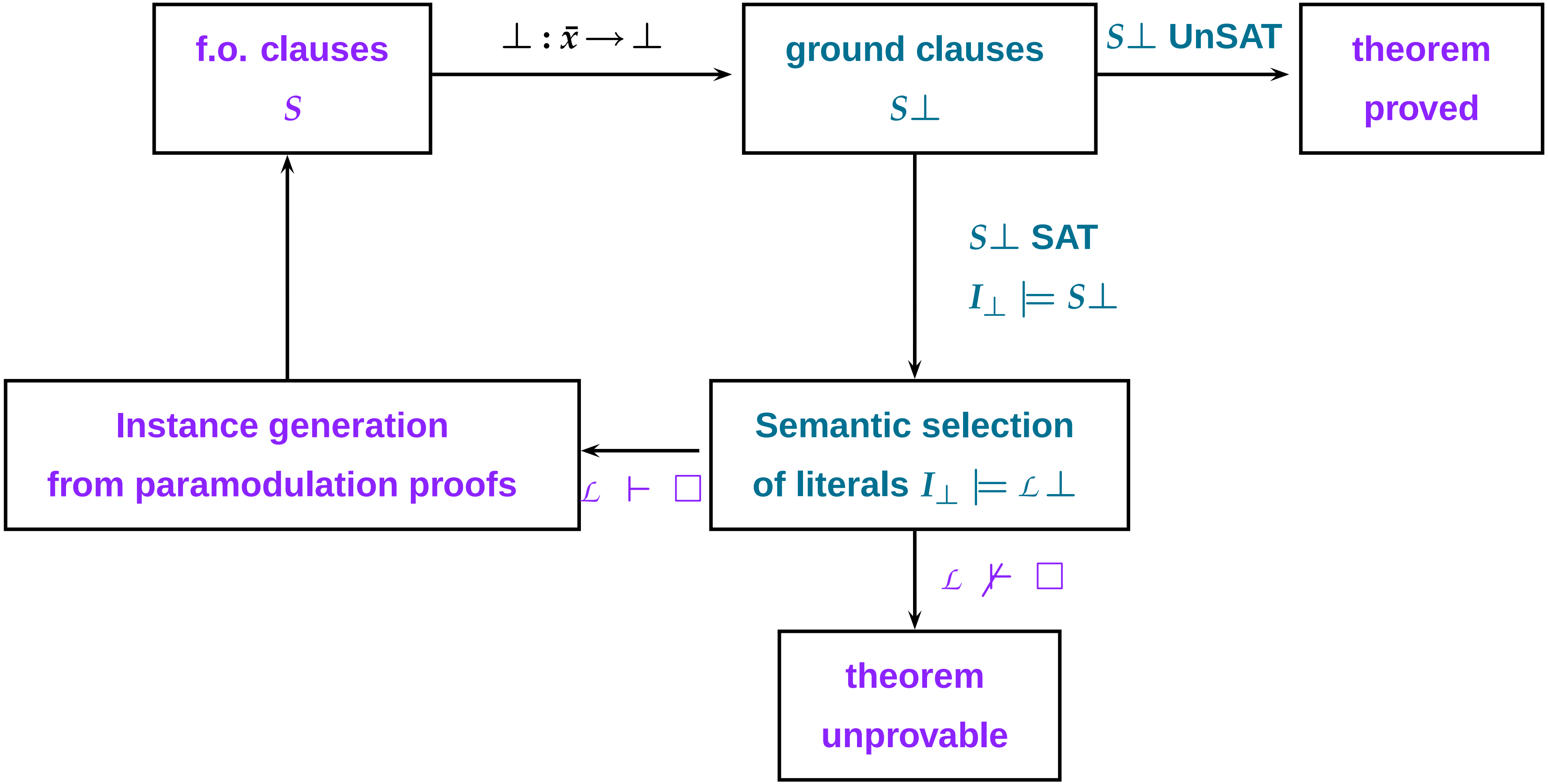
Inst-Gen and Equational Reasoning (II)



Inst-Gen and Equational Reasoning (II)



Inst-Gen and Equational Reasoning (II)



Assessment of Equality Handling Methods

- **Axiomatic Equality Handling**
  - **+ Can be used without modification of prover**
  - **- Incompatible with orderings, hopelessly inefficient**

Assessment of Equality Handling Methods

- **Axiomatic Equality Handling**
  - **+ Can be used without modification of prover**
  - **- Incompatible with orderings, hopelessly inefficient**
- **Eq-Linking**
  - **+ Proven standard technique, compatible with orderings**
  - **- Slightly increases clause lengths**

Assessment of Equality Handling Methods

- Axiomatic Equality Handling
  - + Can be used without modification of prover
  - - Incompatible with orderings, hopelessly inefficient
- Eq-Linking
  - + Proven standard technique, compatible with orderings
  - - Slightly increases clause lengths
- Disagreement Linking
  - + Due to basicness can sometimes detect satisfiability more easily
  - - Incompatible with orderings, creates long clauses

Assessment of Equality Handling Methods

- Axiomatic Equality Handling
  - + Can be used without modification of prover
  - - Incompatible with orderings, hopelessly inefficient
- Eq-Linking
  - + Proven standard technique, compatible with orderings
  - - Slightly increases clause lengths
- Disagreement Linking
  - + Due to basicness can sometimes detect satisfiability more easily
  - - Incompatible with orderings, creates long clauses
- Inst-Gen Equational Instance Generation [Ganzinger and Korovin, 2004]
  - + Maintains separation of first-order and SAT part
  - + Good redundancy elimination, clauses do not grow in length
  - - Not implemented

Implementations and Techniques

Available Implementations

- **Some implementations of instantiation based methods have been realised**

Available Implementations

- Some implementations of instantiation based methods have been realised
  - CLIN-S: ancient implementation of Hyperlinking

Available Implementations

- Some implementations of instantiation based methods have been realised
  - CLIN-S: ancient implementation of Hyperlinking
  - LINUS: hyperlinking with unit support (obsolete)

Available Implementations

- Some implementations of instantiation based methods have been realised
  - CLIN-S: ancient implementation of Hyperlinking
  - LINUS: hyperlinking with unit support (obsolete)
  - PPI: to our knowledge prototypical implementation

Available Implementations

- Some implementations of instantiation based methods have been realised
  - CLIN-S: ancient implementation of Hyperlinking
  - LINUS: hyperlinking with unit support (obsolete)
  - PPI: to our knowledge prototypical implementation
  - OHSL-U: Ordered Semantic Hyperlinking by Plaisted et al.

Available Implementations

- Some implementations of instantiation based methods have been realised
  - CLIN-S: ancient implementation of Hyperlinking
  - LINUS: hyperlinking with unit support (obsolete)
  - PPI: to our knowledge prototypical implementation
  - OHSL-U: Ordered Semantic Hyperlinking by Plaisted et al.
  - DARWIN: Model Evolution prover written in OCaml

Available Implementations

- Some implementations of instantiation based methods have been realised
  - CLIN-S: ancient implementation of Hyperlinking
  - LINUS: hyperlinking with unit support (obsolete)
  - PPI: to our knowledge prototypical implementation
  - OHSL-U: Ordered Semantic Hyperlinking by Plaisted et al.
  - DARWIN: Model Evolution prover written in OCaml
  - DCTP: disconnection calculus tableau prover written in Scheme
- Of the implementations named above, DARWIN and DCTP participated in CASC-J2 (and CASC-20).
- Unfortunately, no implementation is available yet for Inst-Gen

Model Evolution - Darwin's Proof Procedure (I)

```
function darwin S
  // input: a clause set S
  // output: either "unsatisfiable"
  //          or a set of literals encoding a model of S
  let Context = ∅    // set of literals
  let L = ¬v  // (pseudo) literal
                    // Context ∪ {L} is the current context
  let Candidates = set of assert literals consisting of the
                          unit clauses in S
  try me(S, Context, L, Candidates)
  catch CLOSED -> "unsatisfiable"
```

$Candidates$: the literals eligible for application of assert or of split

Model Evolution - Darwin's Proof Procedure (II)

function $\mathit{me}(S, \mathit{Context}, K, \mathit{Candidates})$
  let $\mathit{Candidates}' =$
      $\mathit{add\_new\_candidates}(S, \mathit{Context}, K, \mathit{Candidates})$
  let $S' = S$ simplified by ***Subsume*** and ***Resolve***
  let $\mathit{Context}' = \mathit{Context} \cup \{K\}$ simplified by ***Compact***
  if $\mathit{Candidates}' = \emptyset$ **then** $\mathit{Context}'$ // Got a model of S'
  else
    let $L = \mathit{select\_best}(\mathit{Candidates}', \mathit{Context}')$
    if L is an assert literal **then**
      $\mathit{me}(S', \mathit{Context}', L, \mathit{Candidates}' \setminus \{L\})$ // assert L
    else
      try
        $\mathit{me}(S', \mathit{Context}', L, \mathit{Candidates}' \setminus \{L\})$ // left split on L
      catch CLOSED ->
        $\mathit{me}(S', \mathit{Context}', \overline{L}^{\mathrm{sko}}, \mathit{Candidates}' \setminus \{L\})$ // right split on L

Model Evolution - Darwin's Proof Procedure (III)

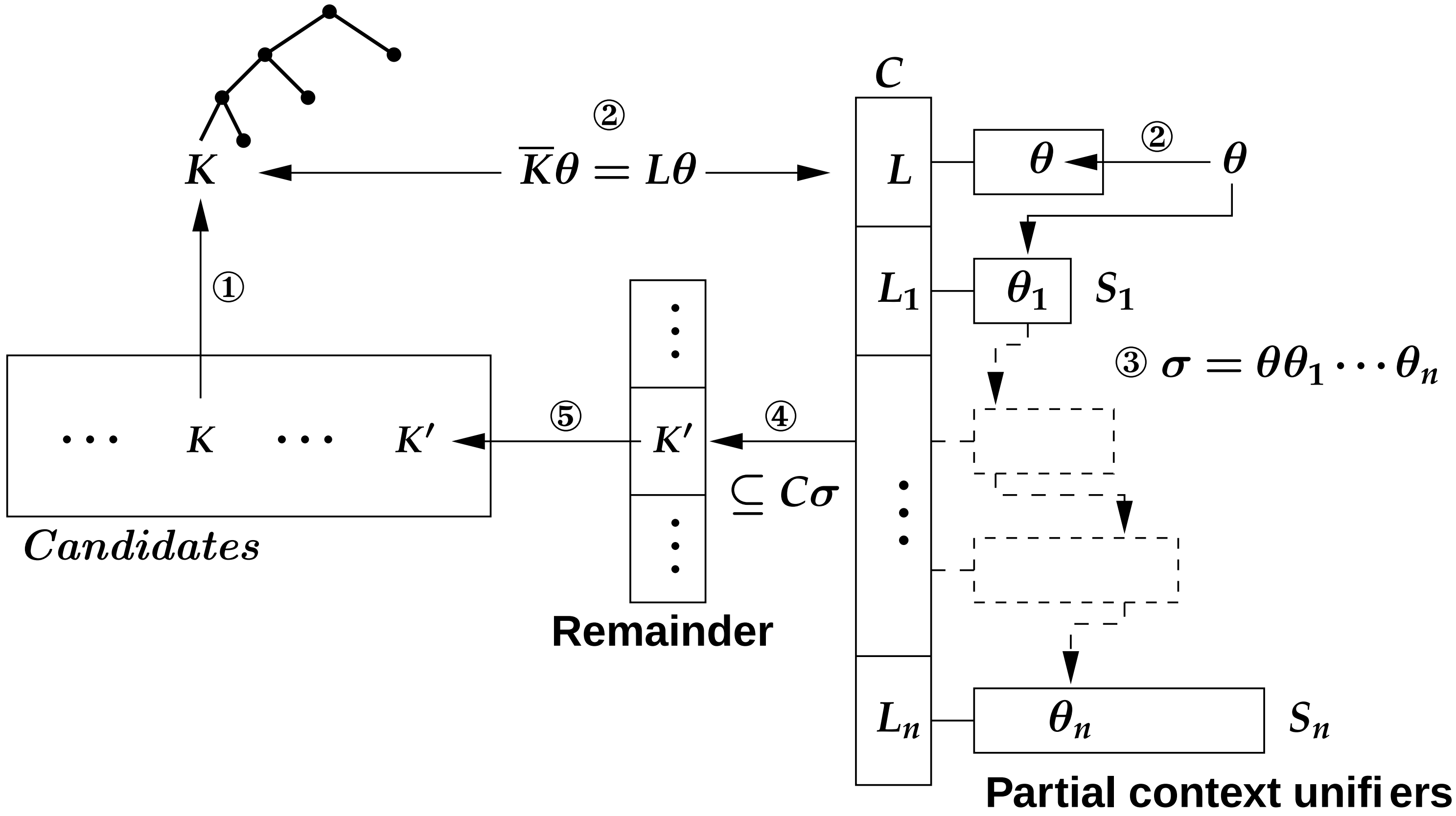
function ***add_new_candidates***(S, $Context$, L, $Candidates$)
  adds to $Candidates$ all assert literals from context unifiers involving L
  and one split literal from each remainder of a context unifier involving L
  raises the exception CLOSED if it finds a closing context unifier

Similar to semi-naïve evaluation of database rules (delta-iteration).

function ***select_best***($Candidates$, $Context$)
  returns the best assert or split literal in $Candidates$

To make select_best good and efficient, *all* theoretically required remainders are kept in store. See next slide.

Computing Remainders and Candidates [Baumgartner e



Partial context unifier: mgu of clause literal and context literal

① add literal K to context – ② compute all partial context unifiers θ of K and clause literals, and store with clause literals – ③ compute all context unifiers involving θ – ④ determine all remainders – ⑤ select K' from remainder and add to candidates (don't care nondeterminism)

Selection Heuristics for New Candidates

In decreasing preferrence:

1. **Universality** (**x - universally quantified; u - schematic variable**)

   $P(x)$ **is better than** $P(u)$

2. **Remainder Size**

   $P(a)$ **is better than** $P(b) \vee Q(b)$

3. **Term Weight**

   $P(a)$ **is better than** $P(f(a))$

4. **Generation**

   Prefer literals from remainders derived from elder context literals

   Rationale: prefer literals close to the original clause set

The Main Loop of DCTP

procedure disconnect($clauses$ **)**

select_initial_path;

create_links($initial\_path$ **);** $start\_sg$ **:= last(** $initial\_path$ **);**

solve_subgoal($start\_sg$, $links$, $initial\_path$ **);**

print("Proof");

The Main Loop of DCTP

```
procedure disconnect( clauses )
    select_initial_path;
    create_links( initial_path ); start_sg := last( initial_path );
    solve_subgoal( start_sg, links, initial_path );
    print( "Proof");

procedure solve_subgoal( sg, links, path )
    if ( ¬ forall_closed( sg ) ) then
      create_new_links( sg );
      if ( apply_linking_step( links ) ) then
        foreach new_sg ∈ ( new_subgoals )
                solve_subgoal( new_sg, links, initial_path ∪ sg );
        end
      else
        print( "Saturation state reached" ); stop;
      endif;
    endif;
```

Picking Up SAT Techniques

Merely a summary of what has been said before

Universal Variables and Unit Propagation

- **Picked up in OSHL, DCTP and ME with varying realization**

Lemma Generation (Learning in SAT)

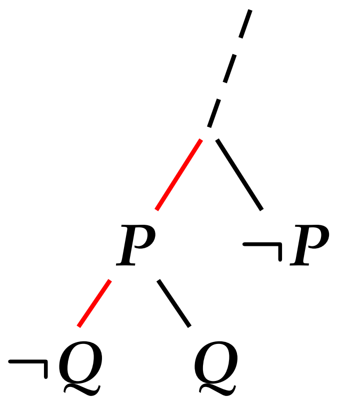
- **Local unit lemmas in DCTP**
- **Global lemma possible in ME (work in progress)**
  In DPLL: lemma clause determined from resolution derivation associated to closed subtree – idea lifts to ME

Other

- **Dependency directed backtracking (backjumping, tableau pruning): a must for any serious prover. . .**
- **DPLL splitting heuristics, randomized restarts – unexplored**

Unit Propagation in SAT

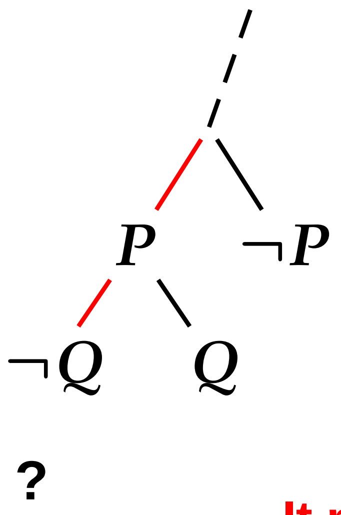
- **Unit propagation is a fundamental technique for efficient SAT proving**
- **Main technical motivation for Model Evolution calculus (see Part I)**



The current open branch
is indicated in red

Unit Propagation in SAT

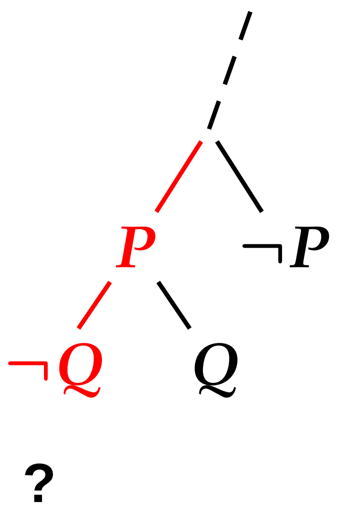
- **Unit propagation is a fundamental technique for efficient SAT proving**
- **Main technical motivation for Model Evolution calculus (see Part I)**



It must be decided over which variable to branch next

Unit Propagation in SAT

- **Unit propagation is a fundamental technique for efficient SAT proving**
- **Main technical motivation for Model Evolution calculus (see Part I)**



Input clauses

$\vdots$

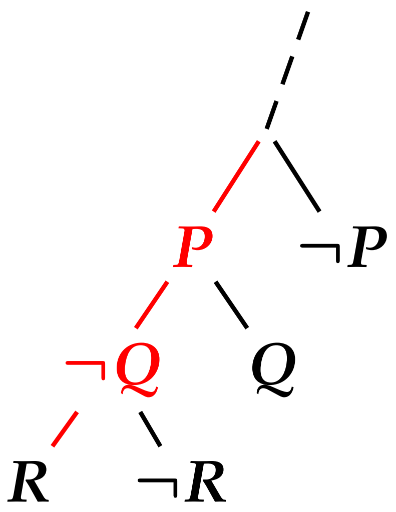
$\neg P \vee Q \vee \neg R$

$\vdots$

The path context is used
to count down the length of input clauses

Unit Propagation in SAT

- Unit propagation is a fundamental technique for efficient SAT proving
- Main technical motivation for Model Evolution calculus (see Part I)



Input clauses

$$\vdots$$

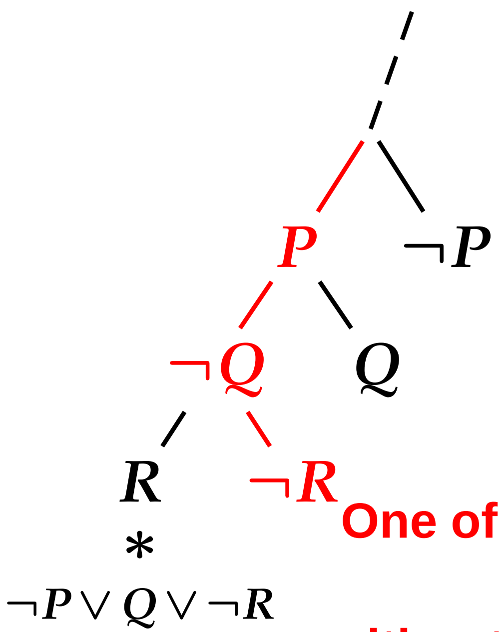
$$\neg P \vee Q \vee \boxed{\neg R}$$

$$\vdots$$

In one input clause
only one unmatched literal remains
This literal is used for the next branching

Unit Propagation in SAT

- Unit propagation is a fundamental technique for efficient SAT proving
- Main technical motivation for Model Evolution calculus (see Part I)



Input clauses

$$\vdots$$

$$\neg P \vee Q \vee \neg R$$

$$\vdots$$

One of the new branches can immediately be closed

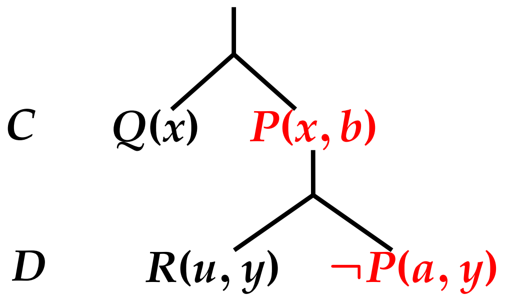
The proof search continues effectively
without branching and with an extended path context

Unit Propagation in Disconnection Tableaux

- **Concept not fully applicable to DC: instantiation influences closure**

Unit Propagation in Disconnection Tableaux

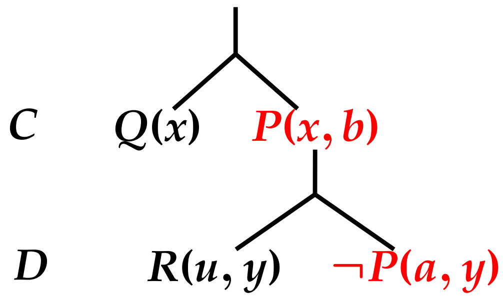
- **Concept not fully applicable to DC: instantiation influences closure**
- **Alternative: count down links instead of clauses [Stenz, 2005]**



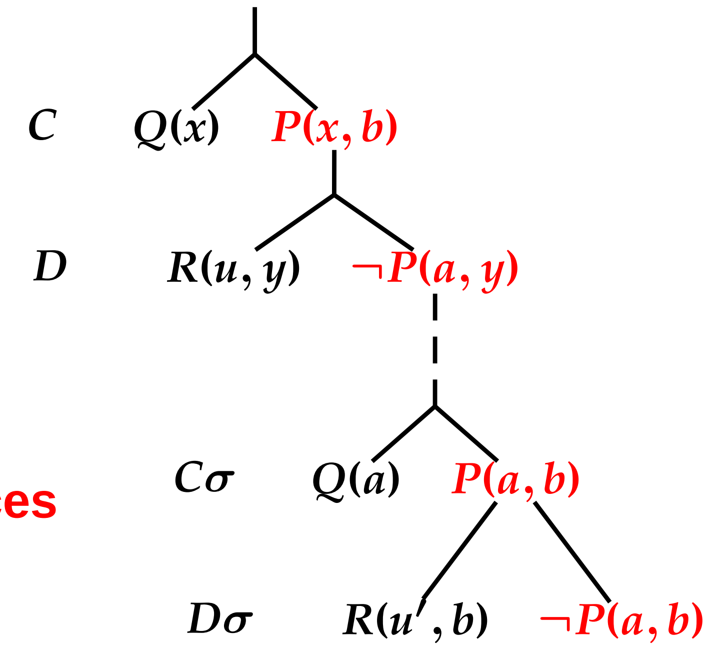
Link: potentially complementary literals on path

Unit Propagation in Disconnection Tableaux

- **Concept not fully applicable to DC: instantiation influences closure**
- **Alternative: count down links instead of clauses [Stenz, 2005]**



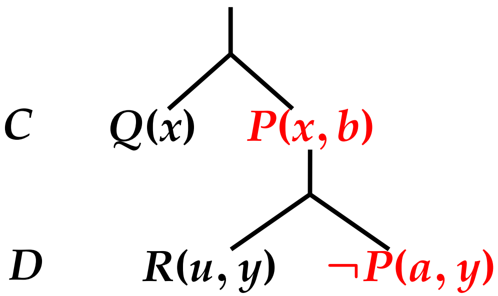
$\Rightarrow$

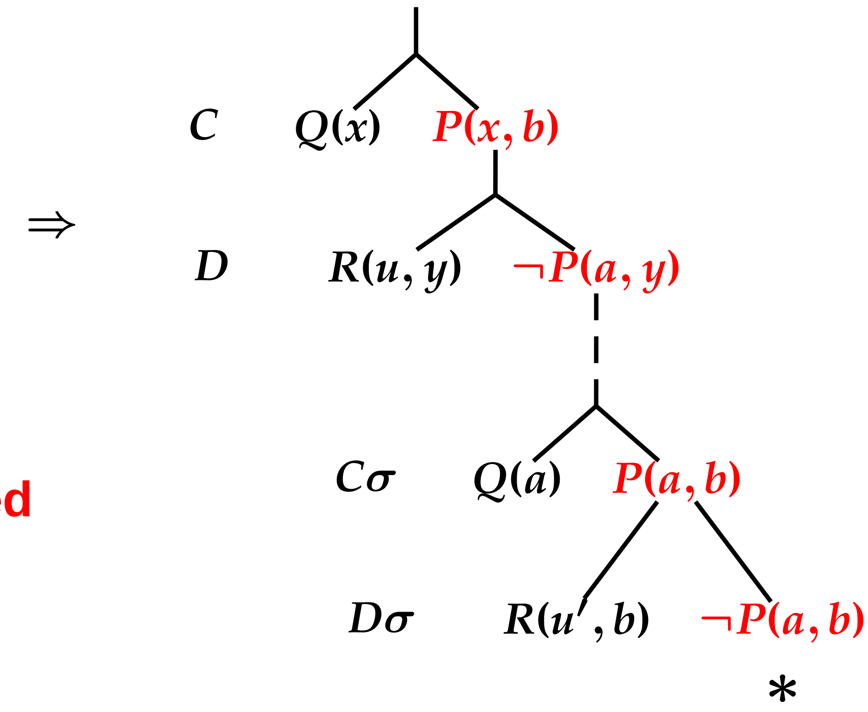


application of link creates linking instances

Unit Propagation in Disconnection Tableaux

- **Concept not fully applicable to DC: instantiation influences closure**
- **Alternative: count down links instead of clauses [Stenz, 2005]**

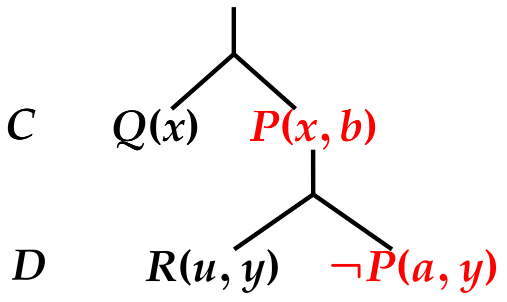




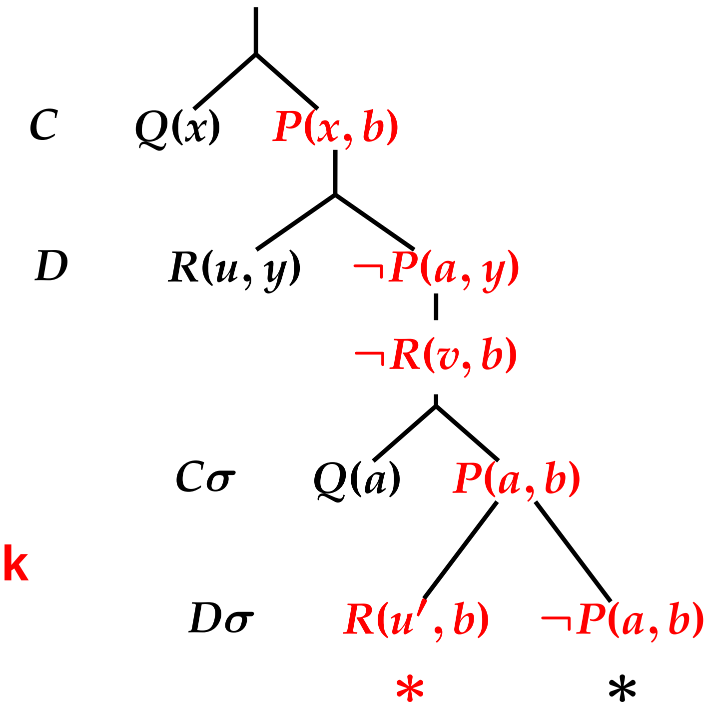
new open paths are created

Unit Propagation in Disconnection Tableaux

- **Concept not fully applicable to DC: instantiation influences closure**
- **Alternative: count down links instead of clauses [Stenz, 2005]**



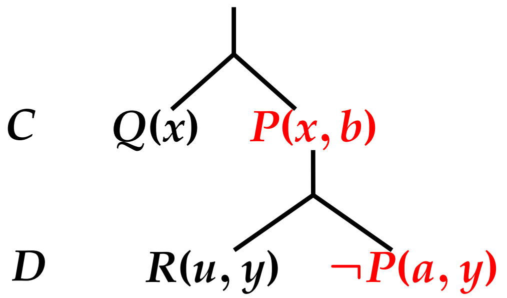
$\Rightarrow$



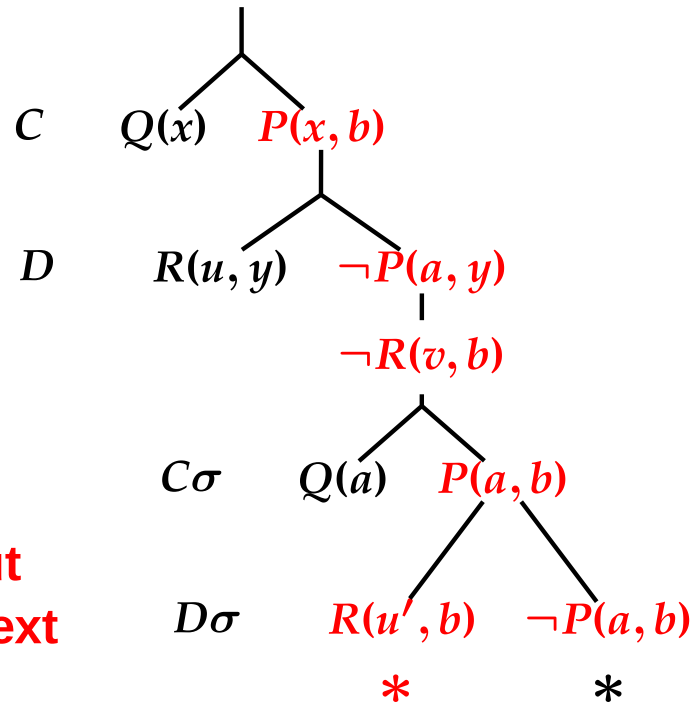
path context is used
to count down open paths created by link

Unit Propagation in Disconnection Tableaux

- **Concept not fully applicable to DC: instantiation influences closure**
- **Alternative: count down links instead of clauses [Stenz, 2005]**



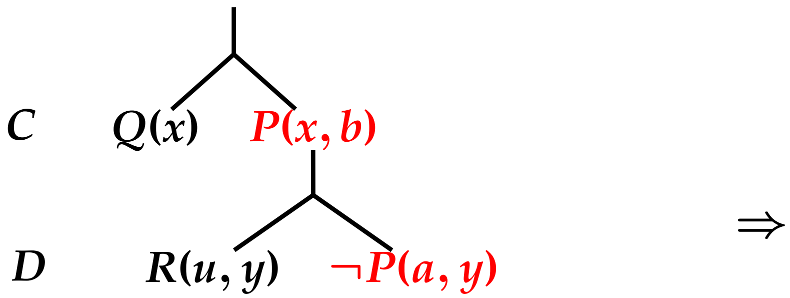
$\Rightarrow$

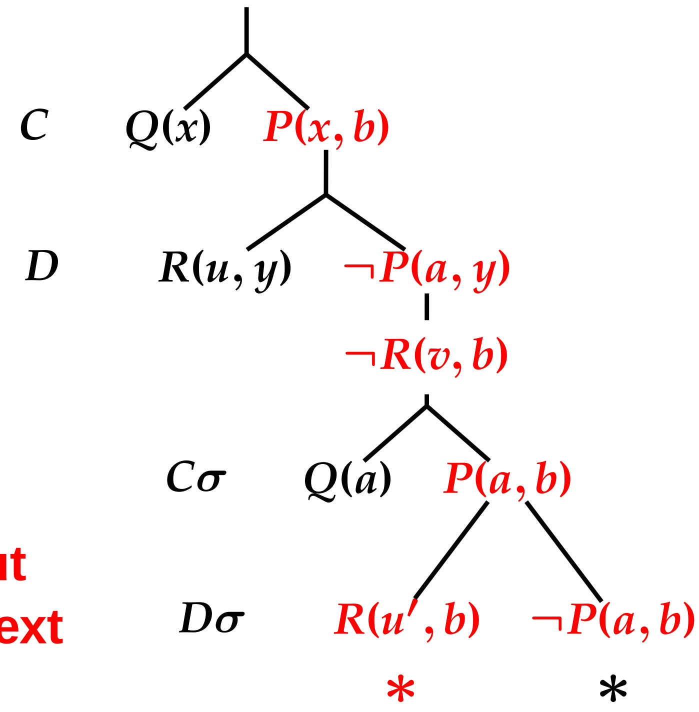


only one new path remains open
proof search continues effectively without branching and with an extended path context

Unit Propagation in Disconnection Tableaux

- **Concept not fully applicable to DC: instantiation influences closure**
- **Alternative: count down links instead of clauses [Stenz, 2005]**





only one new path remains open
proof search continues effectively without
branching and with an extended path context

- **Method need not terminate due to new links by new instances**
- **Selection heuristic instead of deciding strategy**

Conclusions

- **Instance Based Methods provide a new angle to tackle problems**

Conclusions

- Instance Based Methods provide a new angle to tackle problems
- Two-level methods able to capitalise on successful SAT technology

Conclusions

- **Instance Based Methods provide a new angle to tackle problems**
- **Two-level methods able to capitalise on successful SAT technology**
- **Single-level methods successful in their own right**

Conclusions

- Instance Based Methods provide a new angle to tackle problems
- Two-level methods able to capitalise on successful SAT technology
- Single-level methods successful in their own right
- Some SAT techniques are liftable to first-order

Conclusions

- Instance Based Methods provide a new angle to tackle problems
- Two-level methods able to capitalise on successful SAT technology
- Single-level methods successful in their own right
- Some SAT techniques are liftable to first-order
- Possible topics for future research
  - Incorporating theory decision procedures
  - Deciding interesting classes of first-order logic
  - Comparing calculi (e.g. stepwise simulation or wrt. instance sets)
  - Improving implementations (more SAT techniques, heuristics, data structures)