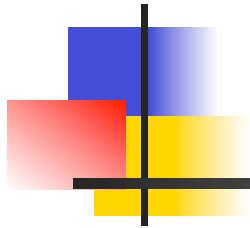


Proof Theory of FOL

Conversion into CNF & SLD-Resolution



Michael Leuschel

Softwaretechnik und Programmiersprachen

Recap: Resolution Principle

1. select literals

1.) $\forall X$ umbrella(X) \vee \neg rains(X)

2.) rains(london)

2. no renaming required

3. mgu $\theta = \{X / \text{london}\}$

1'.) umbrella(london) \vee \neg rains(london)

2'.) rains(london)

4. apply θ

5. resolution

1'+2') umbrella(london)



Overview



I. Algorithm for transforming into CNF

II. Controlling Resolution: SLD



Moving to CNF

$\{\neg \text{rains}(\mathbf{X}) \vee \text{umbrella}(\mathbf{X}), \text{rains}(\mathbf{Z})\}$

- Examples for CNF:

- $(\neg \text{rains} \vee \text{umbrella}) \wedge (\text{rains})$
- $\forall \mathbf{X} \forall \mathbf{Z} (\neg \text{rains}(\mathbf{X}) \vee \text{umbrella}(\mathbf{X})) \wedge (\text{rains}(\mathbf{Z}))$

- Steps:

- Eliminate Implications (\rightarrow)
- Move negations (\neg) down to atomic formulas
- **Eliminate existential quantifiers (\exists)**
- **Rename variables and move \forall 's to the left**
- Move disjunctions (\vee) down to literals
- (Eliminate conjunctions and universal quantifiers)



Example

- Eliminate Implications (\rightarrow)
- Move negations (\neg) down to atomic formulas
- **Eliminate existential quantifiers (\exists)**
- **Rename variables and move \forall 's to the left**
- Move disjunctions (\vee) down to literals

$$\neg \exists X (p(X) \rightarrow \exists Y q(X, Y))$$

Rename Variables and Move \forall 's left

- Example

- $\forall X (\neg \text{rains}(X) \vee \text{umbrella}(X)) \wedge \forall X (\text{rains}(X))$

- Rename quantified variables (if necessary):

- $\forall X (\neg \text{rains}(X) \vee \text{umbrella}(X)) \wedge \forall Z (\text{rains}(Z))$

- Move all quantifiers to the left

- $\forall X \forall Z (\neg \text{rains}(X) \vee \text{umbrella}(X)) \wedge (\text{rains}(Z))$
(no problem as all quantifiers use different variables)



Eliminate Existential Quantifiers

- Example without \forall :

$\exists X \text{ rains}(X)$

- Solution:

- Invent a new constant: **glasgow** (or **cst_new_1** or ...)
- Replace quantified variable by the constant:
 $\text{rains}(\text{glasgow})$
- Called a **Skolem constant**
- Changes the models **but not unsatisfiability** !
 $\text{rains}(\text{glasgow}) \Rightarrow \exists X \text{ rains}(X)$, but not vice-versa
 $\text{rains}(\text{glasgow})$ has a model iff $\exists X \text{ rains}(X)$ has



Eliminate Existential Quantifiers II

- Example

$\forall Z \exists X \text{ father}(X, Z)$

- Solution: ??

$\forall Z \text{ father}(\text{ramses}, Z)$

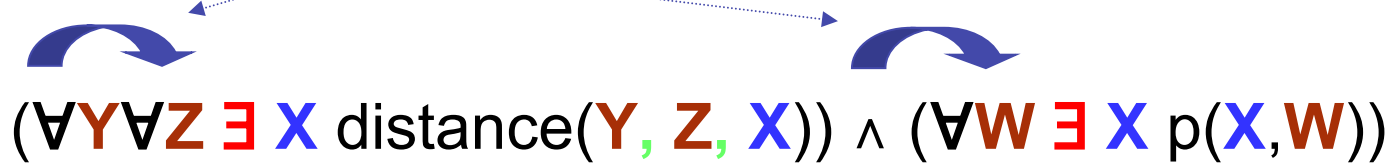
- No: not everybody has the same person as father !
- What is the difference with before ?
- X is in the “scope” of $\forall Z \Rightarrow X$ depends on Z !
- Solution: new **Skolem Function**

$\forall Z \text{ father}(f_1(Z), Z)$

Eliminate Existential Quantifiers III

- General Solution:

- One argument for every universally quantified variable in “scope”



$$(\forall Y \forall Z \exists X \text{ distance}(Y, Z, X)) \wedge (\forall W \exists X p(X, W))$$

- Transformed into:

$$(\forall Y \forall Z \text{ distance}(Y, Z, d(Y, Z))) \wedge (\forall W p(f(W), W))$$

- Preserves **unsatisfiability**



Overview

I. Algorithm for transforming into CNF

 **II. Controlling Resolution: SLD**



Controlling Resolution

- **Unification:**
 - Deterministic
 - No backtracking required
- **Resolution step** after unification
 - Deterministic
- **Select literals** to be unified and resolved:
 - **Non-deterministic choice !**



Example

- 1.) umbrella(X) \vee \neg rains(X)
- 2.) rains(london)
- 3.) rains(singapore)
- 4.) \neg umbrella(Z)
- 5.) rains(edinburgh) \vee rains(glasgow)

- 6.) umbrella(london)
- 7.) umbrella(singapore)
- 8.) \neg rains(Z)
- 9.) umbrella(edinburgh) \vee rains(glasgow)
- 10.) umbrella(glasgow) \vee rains(edinburgh)



Explosion Problem

- **Exponential-Explosion**
 - Many choices for the literals
 - Growth of the number of clauses
- Not suitable for a Programming Language !

- Solution: **Linear Resolution**
 - Restrict to Horn Clauses and 1 Denial
 - Always resolve 1 literal from the denial
 - One can forget earlier denials



Horn Clauses

- **Horn clause** if at most 1 positive literal
 - **Program clause** if exactly 1 positive literal (the head)
 - $\text{knows_logic}(X) \vee \neg \text{logician}(X)$
 - $\text{knows_logic}(X) \leftarrow \text{logician}(X)$
 - $\text{knows_logic}(X) \text{ :- } \text{logician}(X) .$ (in Prolog)
 - **Fact** if 1 positive literal and no negative literal
 - $\text{logician}(\text{peter})$
 - $\text{logician}(\text{peter}) .$ (in Prolog)
 - **Denial** if no positive literal
 - $\neg \text{knows_logic}(X)$ also written: $\leftarrow \text{knows_logic}(X)$
 - $\text{?-knows_logic}(X) .$ (in Prolog)



Example

1) **knows_logic(X)** \vee
 \neg good_student(X) \vee
 \neg teacher(Y,X) \vee
 \neg logician(Y)

2) **good_student(tom)**

3) **logician(peter)**

4) **teacher(peter,tom)**

Program

=

09/05/2011 **Horn Clauses**

- \neg knows_logic(Z)
- \neg good_student(X) \vee
 \neg teacher(Y,X) \vee
 \neg logician(Y)
- \neg teacher(Y,tom) \vee
 \neg logician(Y)
- \neg logician(peter)
- \square



SLD-Resolution

- **S**election-rule driven **L**inear Resolution for **D**efinite Clauses
- Selection-rule
 - In a denial: **selects** a literal to be resolved
- Linear Resolution
 - Derive new denial, **forget** old denial
- Definite Clauses:
 - Limited to (**Definite**) Program Clauses
 - (Normal clause: negation allowed in body)

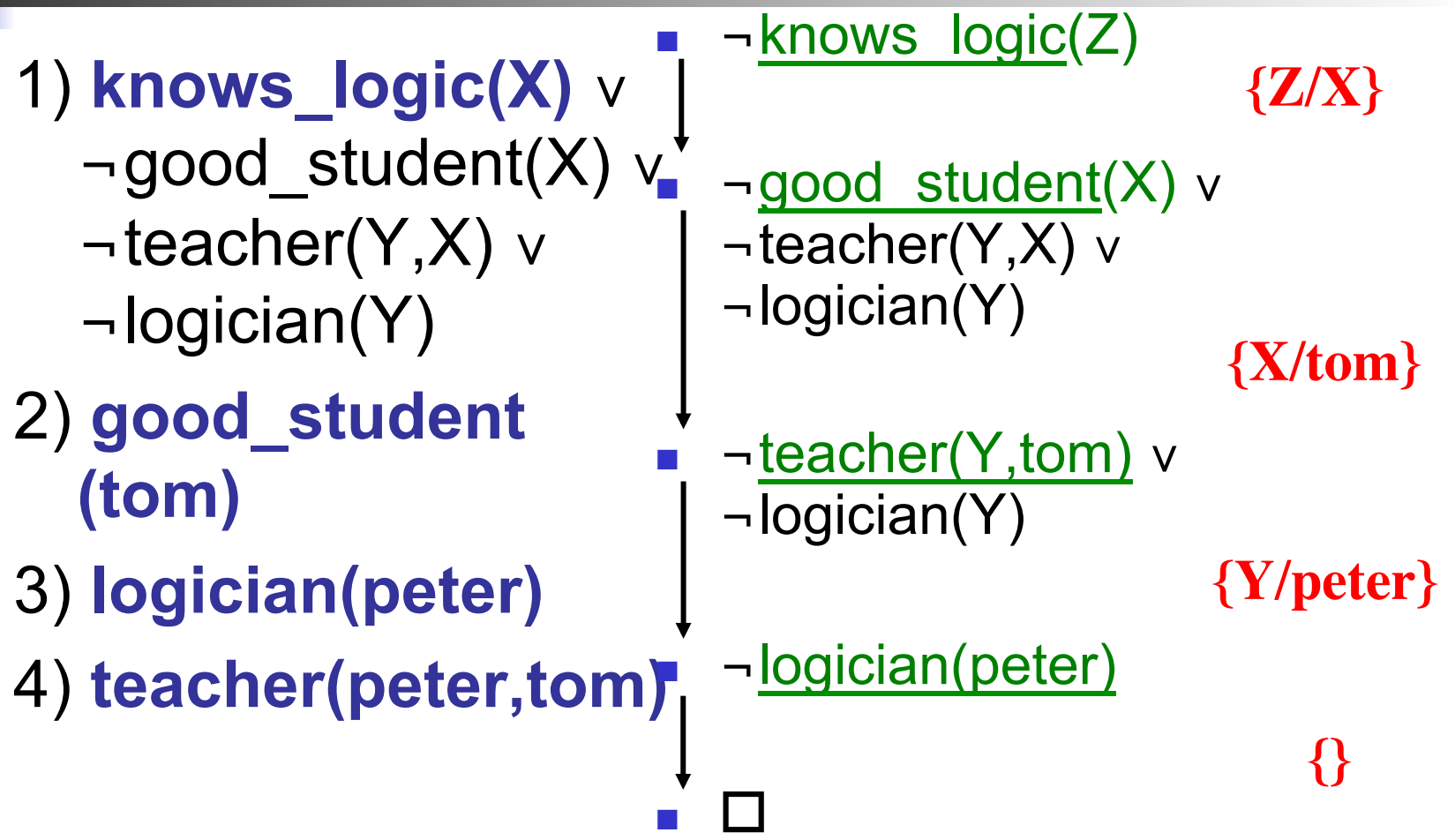


Terminology

- SLD-Derivation for $P \cup \{\leftarrow G\}$
 - Sequence of goals and mgu's, at every step:
 - Apply **selection rule** on current goal (very first goal: G)
 - **Unify** selected literal with the head of a program clause
 - Derive the next goal by **resolution**
- SLD-Refutation for $P \cup \{\leftarrow G\}$
 - SLD-Derivation ending in \square
- Computed answer for $P \cup \{\leftarrow G\}$
 - Composition of mgu's of a SLD-refutation, restricted to variables in G

Example Revisited

G



Computed answer = {Z/tom}



Soundness and Completeness

Let X_1, \dots, X_n be the variables in the denial $\leftarrow G\theta$

■ Soundness

- If θ is a computed answer for for $P \cup \{\leftarrow G\}$ then
 $P \Rightarrow \forall X_1, \dots, X_n (G\theta)$

■ Completeness

- If $P \Rightarrow \forall X_1, \dots, X_n (G\theta)$ then
there exists a computed answer for $P \cup \{\leftarrow G\}$
which is more general than or equal to θ

Backtracking

1) **knows_logic(X)** v
¬good_student(X) v
¬teacher(Y,X) v
¬logician(Y)

2) **good_student(jane)**

3) **good_student(tom)**

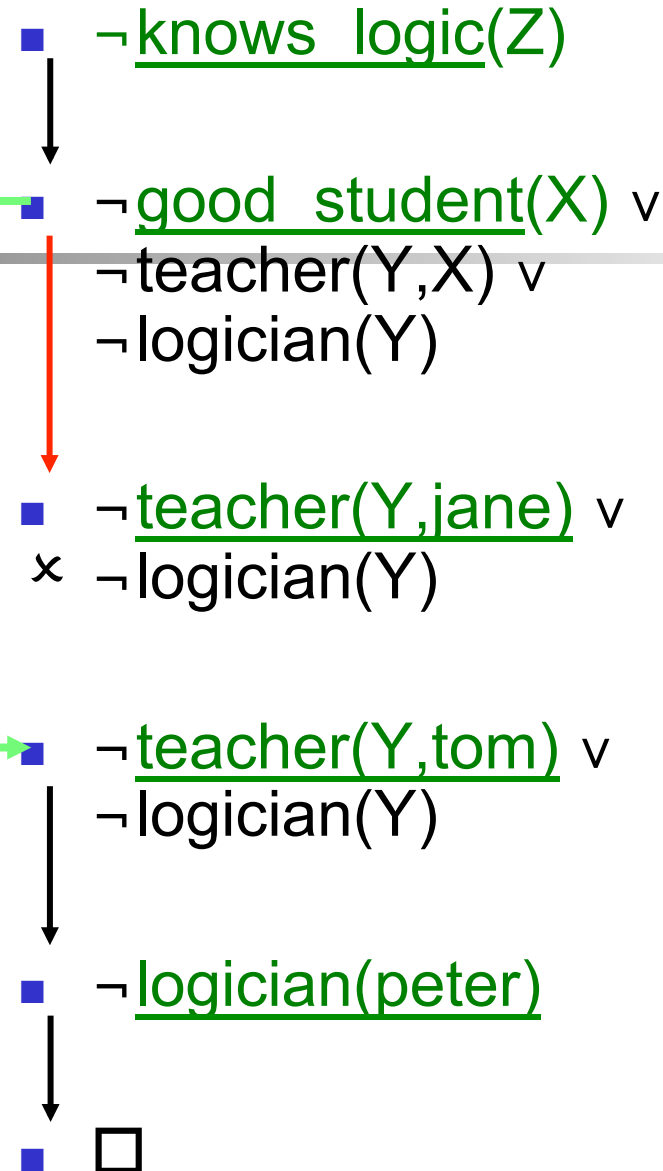
4) **logician(peter)**

5) **teacher(peter,tom)**

Backtracking required !

Try to resolve with another clause

09/05/2011



Backtracking II

1) **knows_logic(X)** v
¬good_student(X) v
¬teacher(Y,X) v
¬logician(Y)

2) **good_student(jane)**

3) **good_student(tim)**

4) **logician(peter)**

5) **teacher(peter,tom)**

■ ¬knows_logic(Z)

■ ¬good_student(X) v

■ ¬teacher(Y,X) v

■ ¬logician(Y)

■ ¬teacher(Y,jane) v

x ¬logician(Y)

■ ¬teacher(Y,tim) v

x ¬logician(Y)

■ ¬good_student(X) v

■ ¬teacher(Y,X) v

■ ¬logician(Y)

Backtracking actually **not** required !
Independence of the selection rule !

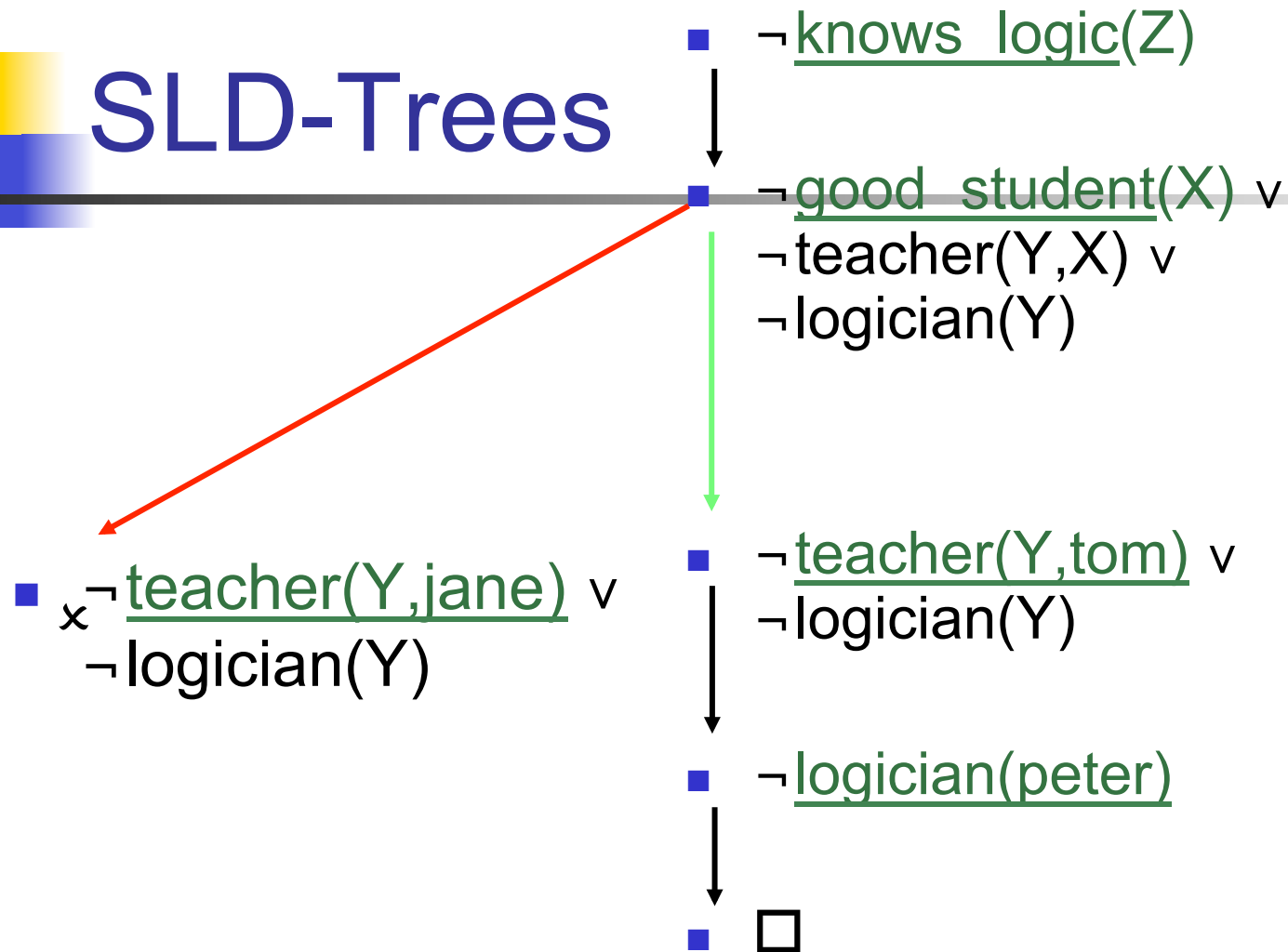


SLD-Trees: Definition

- Root
 - Initial query
- Leaves:
 - Success \square or
 - If no head of clause unifies with selected atom
- Ever non-success node:
 - Selected atom is underlined
 - Children are all resolvents



SLD-Trees



Prolog: explores this tree Depth-First, always selects leftmost literal



Summary

- How to transform FOL into CNF
 - Skolem constants and functions
 - More in the tutorials

- SLD-Resolution
 - Horn Clauses, Denials
 - Principle of linear resolution
 - Soundness and Completeness



Summary of Theory Part

- Propositional & First-Order Logic
 - How to do proof by resolution & refutation
- Prolog Theory
 - A Prolog program corresponds to a FOL theory
 - How Prolog execution corresponds to proof
- Prolog Practice
 - How to write simple Prolog programs for matching & parsing, search, expert systems & reasoning,...



What you haven't seen yet

- Negation as failure & non-monotonic reasoning
- Co-routining
- Constraint Logic Programming



Negation as Failure

- $S = \{ \text{nat}(0), \text{nat}(s(X)) \vee \neg \text{nat}(X) \}$
- $S \Rightarrow \text{nat}(a)$?
 - Proof by refutation: add clause $\neg \text{nat}(a)$
 - unification of $\text{nat}(a)$ with $\text{nat}(0)$ or $\text{nat}(s(X))$ fails
 - (But we don't have $S \Rightarrow \neg \text{nat}(a)$ either)

- Negation as Failure:

- Proving $S \Rightarrow \text{nat}(a)$ has failed
- So, we infer (by NAF rule):
$$S \Rightarrow_{\text{naf}} \neg \text{nat}(a)$$


Prolog:

```
>?-nat(0).  
Yes  
>?-nat(a).  
No  
>
```



Co-routining

- Syntax: **when(condition, Call)**
 - $p(X, Y) :- \mathbf{when}(\text{ground}(X), Y \text{ is } X+1).$
 - Prevents errors
 - Prevents non-termination
 - Makes programs more usable and efficient



CLP

- Classical Prolog
 - $p(a,b)$. same as $p(X,Y) :- X=a, Y=b$.
- Constraint Logic Programming (CLP)
 - CLP(R) - Reals
 - $p(X,Y,Z) :- \{X > Y^2 + Z, Z = 2 * X\}$.
 - CLP(FD) - Finite Domains
 - $p(X,Y) :- X \text{ in } 1..9, Y \text{ in } 2..5, X \neq Y^2$.