

# Programming Systems in Artificial Intelligence

## Introduction

Siegfried Nijssen

23/02/16



**Universiteit  
Leiden**  
The Netherlands

# Programming Paradigms in AI

Two main traditional programming paradigms in AI:

- Logic programming
- Functional programming

# Logic Programming

Developed in 1972-1973 to allow the implementation of intelligent systems that one could ask questions to

Every psychiatrist is a person.

Every person he analyzes is sick.

Jacques is a psychiatrist in Marseille.

Is Jacques a person?

Where is Jacques?

Is Jacques sick?



# Logic Programming

Most popular language is Prolog

Prolog is the basis for:

- Constraint logic programming systems (Eclipse)
- Probabilistic logic programming systems (Problog, Markov logic, PRISM)
- Algebraic logic programming systems (Dyna, aProblog)

# Functional Programming

- Development started in 1958: LISP (LISt Processing)
- Program and data consists of lists that are manipulated using functions
- Expressions perform *symbolic manipulation*
- Most of early AI was symbolic
- LISP was used for instance in:
  - The Dynamic Analysis and Replanning Tool used during the first Gulf War to plan military movements
  - SPIKE, the planning and scheduling application for the Hubble Space Telescope
  - American Express Authorizer's Assistant, checking credit card transactions in the early 1990s

# Functional Programming

- Examples:
  - LISP
  - Scheme
  - Haskell
  - F#
  - Clojure
- Is the basis for
  - Probabilistic functional programming: Church (based on Scheme), PFP (Haskell), Infer.NET (F#), Anglican (Clojure)
  - Functional reactive programming: Observable (F#), Reactive-banana (Haskell)

# Programming Systems in Artificial Intelligence

# Logic & Logic Programming

Siegfried Nijssen

23/02/16



Universiteit  
Leiden  
The Netherlands

# Overview

- Propositional logic
- Predicate logic
- Prolog



# Propositional Logic

- Connectives, in order of increasing precedence:

- $\rightarrow$  : implication
- $\vee$  : or, disjunction
- $\wedge$  : and, conjunction
- $\neg$  : not, negation

- Examples of valid formulas:

$$((a \vee b) \wedge c)$$

( $a, b, c, d$  are called *atoms*)

$$a \vee b \wedge c$$

$$a \vee b \rightarrow c \wedge d$$

$$a \vee b \wedge c \rightarrow d$$

- Implication is right associative:  $a \rightarrow b \rightarrow c \equiv a \rightarrow (b \rightarrow c)$

# Propositional Logic

- “*Interpretation*”, “*truth assignment*”, “*valuation*”, “*possible world*”: synonyms for an assignment of truth values to every atom in a formula
- All interpretations/truth assignments/valuations for  $a \vee b$  are

<b><i>a</i></b>	<b><i>b</i></b>
True	True
True	False
False	True
False	False

- A *model* for a formula is an interpretation that makes the formula true

$$(a \vee b \vee c) \wedge (\neg a \vee b) \wedge \neg b$$

$$b = \text{false}, a = \text{false}, c = \text{true}$$

# Clauses

- Clauses are formulas consisting only of  $\vee$  and  $\neg$

$$p \vee q \vee \neg r$$
$$\neg p \vee \neg q$$

(brackets within a clause are not allowed!)

they can also be written using  $\rightarrow$ ,  $\vee$  (after  $\rightarrow$ ) and  $\wedge$  (before  $\rightarrow$ )

$$r \rightarrow p \vee q$$

$$p \wedge q \rightarrow \perp$$

Clause without positive literal

$$\top \rightarrow p \vee q$$

Clause without negative literal

$$\top \rightarrow \perp$$

Empty clause is considered *false*

an atom or its negation is called a *literal*

# Conjunctive & Disjunctive Normal Form

- A formula is in **conjunctive normal form** if it consists of a conjunction of clauses

$$(p \vee q \vee \neg r) \wedge (p \vee \neg q) \wedge (p \vee r)$$

$$(r \rightarrow p \vee q) \wedge (q \rightarrow p) \wedge (\top \rightarrow p \vee r)$$

- “conjunction of disjunctions”

- A formula is in **disjunctive normal form** if it consists of a disjunction of conjunctions

$$(p \wedge q \wedge \neg r) \vee (p \wedge \neg q) \vee (p \vee r)$$

# Conjunctive & Disjunctive Normal Form

- The transformation from CNF to DNF is exponential

$$(p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge (p_3 \vee q_3) = \begin{aligned} & (p_1 \wedge p_2 \wedge p_3) \vee \\ & (p_1 \wedge p_2 \wedge q_3) \vee \\ & (p_1 \wedge q_2 \wedge p_3) \vee \\ & (p_1 \wedge q_2 \wedge q_3) \vee \\ & (q_1 \wedge p_2 \wedge p_3) \vee \\ & (q_1 \wedge p_2 \wedge q_3) \vee \\ & (q_1 \wedge q_2 \wedge p_3) \vee \\ & (q_1 \wedge q_2 \wedge q_3) \end{aligned}$$

# Conjunctive Normal Form

- Any formula can be written in CNF

$$\begin{aligned}(p \vee q \rightarrow r) \vee (q \rightarrow p) &= \neg(p \vee q) \vee r \vee \neg q \vee p \\&= (\neg p \wedge \neg q) \vee r \vee \neg q \vee p \\&= (\neg p \vee r \vee \neg q \vee p) \\&\quad \wedge (\neg q \vee r \vee \neg q \vee p) \\&= (\neg q \vee r \vee p)\end{aligned}$$

(consequently, any formula can also be written in DNF, but the DNF formula may be exponentially larger)

# Checking Satisfiability of Formulas in CNF

## Example:

solving graph coloring with  $k$  colors by looking for models for a formula in CNF

- for each node  $i$ , create a formula

$$\phi_i = p_{i1} \vee p_{i2} \vee \dots \vee p_{ik}$$

indicating that each node  $i$  must have a color

- for each node  $i$  and different pair of colors  $c_1$  and  $c_2$ , create a formula

$$\phi_{ic_1c_2} = \neg(p_{ic_1} \wedge p_{ic_2}) = \neg p_{ic_1} \vee \neg p_{ic_2}$$

indicating a node may not have more than 1 color

- for each edge, create  $k$  formulas

$$\phi_{ijc} = \neg(p_{ic} \wedge p_{jc}) = \neg p_{ic} \vee \neg p_{jc}$$

indicating that a pair connected nodes  $i$  and  $j$  may not both have color  $c$  at the same time

# Resolution Rule

The resolution rule for clauses:

Given two clauses  $l_1 \vee \dots \vee l_k$  and  $m_1 \vee \dots \vee m_n$   
where  $l_1, \dots, l_k, m_1, \dots, m_n$  represent literals:  
if it holds that  $l_i = \neg m_j$ , then it holds that

$$l_1 \vee \dots \vee l_k, m_1 \vee \dots \vee \dots m_n \vdash_R$$
$$l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n$$

Example:  $p \vee q \vee \neg r, r \vee s \vdash_R p \vee q \vee s$   
 $r \rightarrow p \vee q, r \vee s \vdash_R p \vee q \vee s$



# Definite clauses & Horn clauses

- A **definite clause** is a clause with exactly one positive literal

$$p, q, p \wedge q \rightarrow t$$

- A **horn clause** is a clause with at most one positive literal

$$p, q, p \wedge q \rightarrow t, p \wedge q \rightarrow \perp$$



A clause with one positive literal is called a **fact**

# Resolution on Definite Clauses

- Example:

$$p \leftarrow q \wedge r$$

$$q \leftarrow t$$

$$t$$

$$r$$

- $p \leftarrow q \wedge r, q \leftarrow t \vdash_R p \leftarrow t \wedge r$

$$p \leftarrow t \wedge r, t \vdash_R p \leftarrow r$$

$$p \leftarrow r, r \vdash_R p$$

- “backchaining algorithm”

# Forward chaining for Definite clauses

- The forward chaining algorithm calculates facts that can be entailed from a set of definite clauses

$C$  = initial set of definite clauses

**repeat**

    if there is a clause  $p_1, \dots, p_n \rightarrow q$  in  $C$  where  $p_1, \dots, p_n$  are

        facts in  $C$  **then**

        add fact  $q$  to  $C$

**end if**

**until** no fact could be added

**return** all facts in  $C$



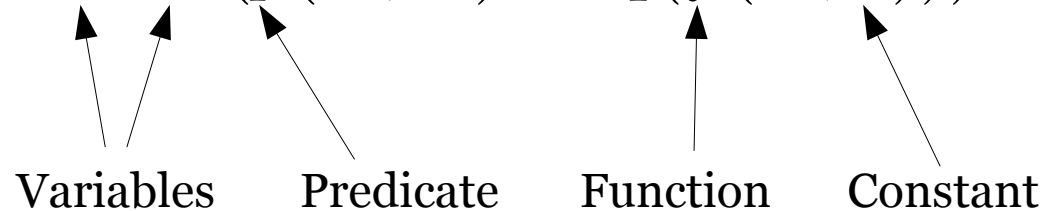
Resolution

# Predicate (First-Order) Logic

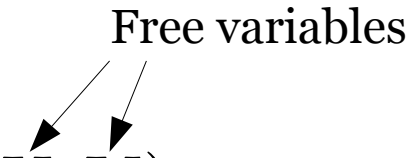
- Extends propositional logic with:
  - Existential quantor  $\exists$
  - Universal quantor  $\forall$
  - Predicates
  - Functions
  - Variables
  - Constants
- Example of a formula in first-order logic:

$$\forall X \forall Y (p(X, Y) \rightarrow q(f(X, a)))$$

Variables      Predicate      Function      Constant



# Predicate (First-Order) Logic

- Precedence:  $\forall, \exists, \neg, \wedge, \vee, \rightarrow$   
 $\forall X p(X) \rightarrow q(X, Y) \equiv (\forall X p(X)) \rightarrow q(X, Y)$   

- Terms: variables, constants, functions applied to terms  
 $X, a, f(X, a), f(f(X, a), a)$
- Atoms: predicates applied to terms  
 $p(X, a), p(f(X, a), a)$
- Literals: atoms or negated atoms  
 $p(X, a), \neg q(X)$

# Predicate (First-Order) Logic

- An interpretation  $\mathcal{A}$  for a formula consists of:
  - A universe of values  $\mathcal{U}_{\mathcal{A}}$
  - For each constant in the formula, a corresponding value from the universe
  - For each function symbol  $f$  of arity  $n$ , a function  $f_{\mathcal{A}} : \mathcal{U}^n \rightarrow \mathcal{U}$
  - For each predicate symbol  $p$  of arity  $n$ , a subset  $p_{\mathcal{A}}$  of  $\mathcal{U}^n$
- Example: an interpretation for  $\forall X (\exists Y p(X, Y) \rightarrow q(f(X)))$  is

$$\mathcal{U}_{\mathcal{A}} = \{1, 2\}$$

$$p_{\mathcal{A}} = \{(1, 2), (2, 2)\}$$

$$q_{\mathcal{A}} = \{2\}$$

$$f_{\mathcal{A}}(1) = 1, f_{\mathcal{A}}(2) = 1$$

→ Not a model for this formula

# Predicate (First-Order) Logic

- A model for a first-order formula is an interpretation that makes the formula true
- A first-order formula is satisfiable if it has a model
- It is undecidable in general whether a formula in first-order logic is satisfiable
- $\varphi \models \psi$  denotes that all models for  $\varphi$  are also models for  $\psi$

# Predicate (First-Order) Logic

- Rewriting first-order logic formulas

$$\neg \exists X p(X) \equiv \forall X \neg p(X)$$

$$\neg \forall X p(X) \equiv \exists X \neg p(X)$$

$$\forall X \forall Y p(X, Y) \equiv \forall Y \forall X p(X, Y)$$

$$\exists X \exists Y p(X, Y) \equiv \exists Y \exists X p(X, Y)$$

$$\exists X \forall Y p(X, Y) \not\equiv \forall Y \exists X p(X, Y) \qquad \exists X \forall Y p(X, Y) \models \forall Y \exists X p(X, Y)$$

- Prenex normal form: all quantors first**

$$\forall X (\exists Y p(X, Y) \rightarrow q(X)) \equiv$$

$$\forall X (\neg \exists Y p(X, Y) \vee q(X)) \equiv$$

$$\forall X (\forall Y \neg p(X, Y) \vee q(X)) \equiv$$

$$\forall X \forall Y (\neg p(X, Y) \vee q(X))$$



# Skolemized Formulas

- Assume we are given a formula  $\varphi$  in prenex normal form
- Then  $\varphi$  is satisfiable if its *skolemized version* is satisfiable
- Essentially, in the skolemized version all existential quantifiers are removed and replaced by
  - new constants (if the existential quantifier is not in the range of a universal quantifier)
  - calls to a new function, with all universally quantified variables as parameters (if the existential quantifier is in the range of universal quantifiers)
- Example:

$$\exists X \forall Y \exists Z (p(X, Y) \rightarrow q(Z)) \Rightarrow \forall Y (p(x, Y) \rightarrow q(z(Y)))$$

# Herbrand Interpretations

- The *herbrand universe* for a skolemized formula  $\varphi$  is the universe of all terms that can be created using the constants and functions in the formula
  - If the formula does not have constants, one constant  $a$  is added in the universe

- Example:

$\forall X \forall Y (p(a, X) \rightarrow p(a, f(Y)))$     Herbrand universe:

$\{a, f(a), f(f(a)), f(f(f(a))), \dots\}$

- The Herbrand base is the set of all atoms that can be created using terms in the Herbrand universe as arguments

Herbrand base for the example:

$p(a, a), p(a, f(a)), p(f(a), a), p(f(a), f(a)), p(f(a), f(f(a))), \dots$

# Herbrand Interpretation & Model

- A Herbrand interpretation for a formula  $\varphi$  is an interpretation in which:
  - The universe is the Herbrand universe
  - The constants are assigned the trivial value in the universe
  - Each function is mapped to the trivial composed value in the universe
  - For each predicate the subset indicating for which arguments the predicate is true, is a subset of the Herbrand base
- Example:

$$\forall X \forall Y (q(a) \wedge q(b) \wedge (p(X, Y) \rightarrow q(X) \wedge q(Y)))$$

- Universe:  $a, b$
- Constant  $a$  takes value  $a$  in the universe, constant  $b$  value  $b$
- The subsets for the predicates are:
  - **For  $p$ :**  $\{p(a, a), p(a, b)\}$
  - **For  $q$ :**  $\{q(a), q(b)\}$

This interpretation is also a model for the formula; such a model is called a *Herbrand model*

# Propositionalization

- If the Skolemized formula does not have functions, the Herbrand base is finite
- Finding a model = Finding a subset of the Herbrand base
- Such a model can be found by turning each element of the Herbrand base in a *propositional atom* and solving a propositional formula corresponding to the first-order logic formula. Idea:  
Replace  $\forall X \varphi(X)$  with  $\varphi(a_1) \wedge \varphi(a_2) \wedge \dots \wedge \varphi(a_n)$  where  $a_1, \dots, a_n$  are all values in the Herbrand universe

- Example:

$$\forall X \forall Y (q(a) \wedge q(b) \wedge (p(X, Y) \rightarrow q(X) \wedge q(Y)))$$

$$\begin{aligned} & q(a) \wedge q(b) \wedge (p(a, a) \rightarrow q(a) \wedge q(a)) \wedge \\ & (p(a, b) \rightarrow q(a) \wedge q(b)) \wedge (p(b, a) \rightarrow q(b) \wedge q(a)) \wedge \\ & (p(b, b) \rightarrow q(b) \wedge q(b)) \end{aligned}$$

First order logic formula  
can be  
seen as template

# Clauses

- A clause is a formula that in prenex normal form can be written as

$$\forall X_1 \cdots \forall X_n \varphi$$

where  $\varphi$  is a disjunction over literals

- **Examples:**

$$\forall X \forall Y (\neg p(X, Y) \vee q(X))$$

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

$$\forall X (\exists Y (p(X, Y) \wedge q(Y)) \rightarrow (q(X) \vee r(X)))$$

- **Sometimes written as**

$$q(X) \leftarrow p(X, Y)$$

$$q(X) \vee r(X) \leftarrow p(X, Y) \wedge q(Y)$$

# Finding Models for CNF Formulas

- If a CNF formula does **not** have functions, it can be propositionalized and finding a model for the first-order CNF formula = finding a model for the propositional CNF formula
- Example:

Formula:  $(p(X) \leftarrow q(X)) \wedge (q(Y) \leftarrow t(Y)) \wedge t(a) \wedge t(b)$

Propositional form:

$$(p(a) \leftarrow q(a)) \wedge (p(b) \leftarrow q(b)) \wedge \\ (q(a) \leftarrow t(a)) \wedge (q(b) \leftarrow t(b)) \wedge t(a) \wedge t(b)$$

Model: make true:  $t(a), t(b), q(a), q(b), p(a), p(b)$

- *Minimal model*: a model that is not a “superset” of another model

# Exercise

- Given the following first-order formula:

$$\exists Z \forall X \forall Y (edge(X, Y) \rightarrow label(X, Z) \wedge label(Y, Z))$$

- Determine its (skolemized) conjunctive normal form
- Determine its Herbrand base
- Propositionalize the formula
- Determine a model for the formula

# Exercise

- Given the following first-order formula:

$$\exists Z \forall X \forall Y (edge(X, Y) \rightarrow label(X, Z) \wedge label(Y, Z)) \\ \wedge label(1, a) \wedge label(2, b) \wedge edge(1, 2)$$

- Determine its (skolemized) conjunctive normal form
- Determine its Herbrand base
- Propositionalize the formula
- Determine a model for the formula
- Determine a *minimal* model for the formula



# Solving Graph Coloring using *Answer Set Programming (Version 1)*

```
node(1). node(2). node(3). edge(1,2). edge(2,3). edge(1,3).
edge(X,Y) :- edge(Y,X).
```

```
1 { coloring(X,green); coloring(X,red); coloring(X,blue) } :- node(X).

:- coloring(X1,C), coloring(X2,C), edge(X1,X2).
:- coloring(X,green), coloring(X,red).
:- coloring(X,green), coloring(X, blue).
:- coloring(X,red), coloring(X,blue).
```

Notation for




---


$$\text{coloring}(X, \text{green}) \vee \text{coloring}(X, \text{red}) \vee \text{coloring}(X, \text{blue}) \leftarrow \text{node}(X)$$

$$\perp \leftarrow \text{coloring}(X1, C) \wedge \text{coloring}(X2, C) \wedge \text{edge}(X1, X2)$$

$$\perp \leftarrow \text{coloring}(X, \text{green}) \wedge \text{coloring}(X, \text{red})$$

$$\perp \leftarrow \text{coloring}(X, \text{green}) \wedge \text{coloring}(X, \text{blue})$$

$$\perp \leftarrow \text{coloring}(X, \text{red}) \wedge \text{coloring}(X, \text{blue})$$

gringo gc.lp | clasp --number 0

# Solving Graph Coloring using *Answer Set Programming (Version 2)*

```
node(1). node(2). node(3). edge(1,2). edge(2,3). Edge(1,3).
edge(X,Y) :- edge(Y,X).
color(green). color(red). color(blue).
```

```
1 { coloring(X,C): color(C) } :- node(X).
```

```
:- coloring(X1,C), coloring(X2,C), edge(X1,X2).
```

```
:- coloring(X,C1), coloring(X,C2), C1 != C2.
```

Notation for



$$\text{coloring}(X, \text{green}) \vee \text{coloring}(X, \text{red}) \vee \text{coloring}(X, \text{blue}) \leftarrow \text{node}(X)$$

$$\perp \leftarrow \text{coloring}(X1, C) \wedge \text{coloring}(X2, C) \wedge \text{edge}(X1, X2)$$

$$\perp \leftarrow \text{coloring}(X, \text{green}) \wedge \text{coloring}(X, \text{red})$$

$$\perp \leftarrow \text{coloring}(X, \text{green}) \wedge \text{coloring}(X, \text{blue})$$

$$\perp \leftarrow \text{coloring}(X, \text{red}) \wedge \text{coloring}(X, \text{blue})$$

# Solving Graph Coloring using *Answer Set Programming (Version 3)*

```
node(1). node(2). node(3). edge(1,2). edge(2,3). edge(1,3).
edge(X,Y) :- edge(Y,X).
color(green). color(red). color(blue).
```

```
1 { coloring(X,C): color(C) } 1 :- node(X).
```

```
:- coloring(X1,C), coloring(X2,C), edge(X1,X2).
```

Notation for




---


$$\text{coloring}(X, \text{green}) \vee \text{coloring}(X, \text{red}) \vee \text{coloring}(X, \text{blue}) \leftarrow \text{node}(X)$$

$$\perp \leftarrow \text{coloring}(X1, C) \wedge \text{coloring}(X2, C) \wedge \text{edge}(X1, X2)$$

$$\perp \leftarrow \text{coloring}(X, \text{green}) \wedge \text{coloring}(X, \text{red})$$

$$\perp \leftarrow \text{coloring}(X, \text{green}) \wedge \text{coloring}(X, \text{blue})$$

$$\perp \leftarrow \text{coloring}(X, \text{red}) \wedge \text{coloring}(X, \text{blue})$$

# Solving Graph Coloring using *Answer Set Programming* – Exercises

```
node(1). node(2). node(3). edge(1,2). edge(2,3). edge(1,3).  
edge(X,Y) :- edge(Y,X).  
color(green). color(red). color(blue).
```

```
1 { coloring(X,C): color(C) } 1 :- node(X).
```

```
:- coloring(X1,C), coloring(X2,C), edge(X1,X2).
```

- Extend the code such that node 1 is ensured to be green.
- Extend the code such that no two neighbors of any given node have the same color

```
0 { coloring(Y,C) : edge(X,Y) } 1 :- node(X), color(C)
```

# Hamilton Cycles in ASP

- Hamilton cycle: visit all nodes exactly once in a cycle

```
node(a). node(b). node(c).  
edge(a,b). edge(b,c).  
edge(X,Y) :- edge(Y,X).
```

```
number(1). number(2). number(3).  
next(1,2). next(2,3). next(3,1).
```

```
1 { step(I,X) : number(I) } 1 :- node(X).  
1 { step(I,X) : node(X) } 1 :- number(I).
```

```
edge(X1,X2) :- step(I1,X1), step(I2,X2), next(I1,I2).
```

# Negation in ASP

- ASP supports negated atoms in the right-hand side of a formula

`:- step(I1,X1), step(I2,X2), next(I1,I2), not edge(X1,X2).`

- ASP gives such rules a special interpretation; intuition:
  - If an atom occurs on the left-hand side of the `:-`, it is allowed to add this atom to the model to make the rule true
  - If an atom occurs negated on the right-hand side of the `:-`, the atom may not be added by this rule; if the atom is not the consequence of some other rule, the rule is false

# Negation in ASP

- A model in ASP is a *stable model* iff the model is also minimal for the grounded (propositional) program in which:
  - All rules for which a right-hand negative literal is false, are removed
  - All right-hand negative literals that are true, are removed
- Example: program

```
:- b, not c.  
b.
```

has as model { b, c }.

This program is reduced to:

b.            (The rule is removed as it has a right-hand literal that is false)

For this program { b, c } is not a minimal model, hence this model is not stable

# Hamilton Cycles in ASP

- `edge(a,c)` is not part of a stable model for the program below:

```
node(a). node(b). node(c).  
edge(a,b). edge(b,c).  
edge(X,Y) :- edge(Y,X).
```

```
number(1). number(2). number(3).  
next(1,2). next(2,3). next(3,1).
```

```
1 { step(I,X) : number(I) } 1 :- node(X).  
1 { step(I,X) : node(X) } 1 :- number(I).
```

```
:- step(I1,X1), step(I2,X2), next(I1,I2), not edge(X1,X2).
```



# Clark's Completion

- An answer set is a model for *Clark's completion* of a model
- Example:

$$a \leftarrow b$$

$$a \leftarrow c$$

Here,  $a$  can be part of the model

Clark's completion:  $a \leftrightarrow (b \vee c)$

Here,  $a$  can not be part of the model.

- In general, Clark's completion for a set of clauses  $a \leftarrow R1, a \leftarrow R2, \dots$  is the formula  $a \leftrightarrow R1 \vee R2 \vee \dots$

# Limitation till Now

- Answer set programming does not support functions
- Up next: Prolog, which supports functions

# Horn, Definite and Goal Clauses

- A **definite clause** is (again) a clause with exactly one positive literal

$$q(X) \leftarrow p(X, Y)$$

~~$$q(X) \vee r(X) \leftarrow p(X, Y) \wedge q(Y)$$~~

- A **Horn clause** is (again) a clause with at most one positive literal

$$\perp \leftarrow p(X, Y)$$

$$q(X) \leftarrow p(X, Y)$$

- A **Goal clause** is a clause with no positive literal

$$\perp \leftarrow p(X, Y)$$

# Substitutions on Clauses

- Given a formula  $\varphi$  a substitution  $\theta$  is a set that maps variables in  $\varphi$  to terms
- A substitution can be applied to a formula, denoted  $\varphi\theta$ , and yields the formula  $\varphi$  in which the variables are replaced with the corresponding new terms
- Examples:

$$\varphi = p(X, Y) \vee q(X) \leftarrow q(Y)$$

$$\theta = \{X \mapsto f(X), Y \mapsto X\}$$

$$\varphi\theta = p(f(X), X) \vee q(f(X)) \leftarrow q(X)$$

$$\varphi = p(X, Y) \wedge q(Y)$$

$$\theta = \{X \mapsto A\}$$

$$\varphi\theta = p(A, Y) \wedge q(Y)$$

# Unification

- A *unifier* is a substitution that makes two atoms identical
  - $p(X, a), p(a, a) \Rightarrow \{X \mapsto a\}$
  - $p(f(a), X), p(Y, a) \Rightarrow \{X \mapsto a, Y \mapsto f(a)\}$
  - $p(f(a), X), p(X, a) \Rightarrow$  impossible
- The *most general unifier* is not unnecessarily large or complex
  - $p(X, Y), p(X, a) \Rightarrow \{Y \mapsto a\} \cancel{\{X \mapsto Z, Y \mapsto a\}}$
  - $p(X, a), p(Y, a) \Rightarrow \{X \mapsto Y\} \cancel{\{X \mapsto a, Y \mapsto a\}}$

# Resolution in First-Order Logic

The resolution rule for clauses:

Given two clauses  $l_1 \vee \dots \vee l_k$  and  $m_1 \vee \dots \vee m_n$  that are standardized apart, i.e., they don't share variables, where  $l_1, \dots, l_k, m_1, \dots, m_n$  represent literals: if it holds that  $l_i\theta = \neg m_j\theta$  for an MGU  $\theta$  for literals  $l_i, m_j$ , then it holds that

$$l_1 \vee \dots \vee l_k, m_1 \vee \dots \vee \dots m_n \vdash_R \\ l_1\theta \vee \dots \vee l_{i-1}\theta \vee l_{i+1}\theta \vee \dots \vee l_k\theta \vee m_1\theta \vee \dots \vee m_{j-1}\theta \vee m_{j+1}\theta \vee \dots \vee m_n\theta$$

Example:  $\neg p(X, Y) \vee q(Y), p(a, b) \vdash_R q(a)$

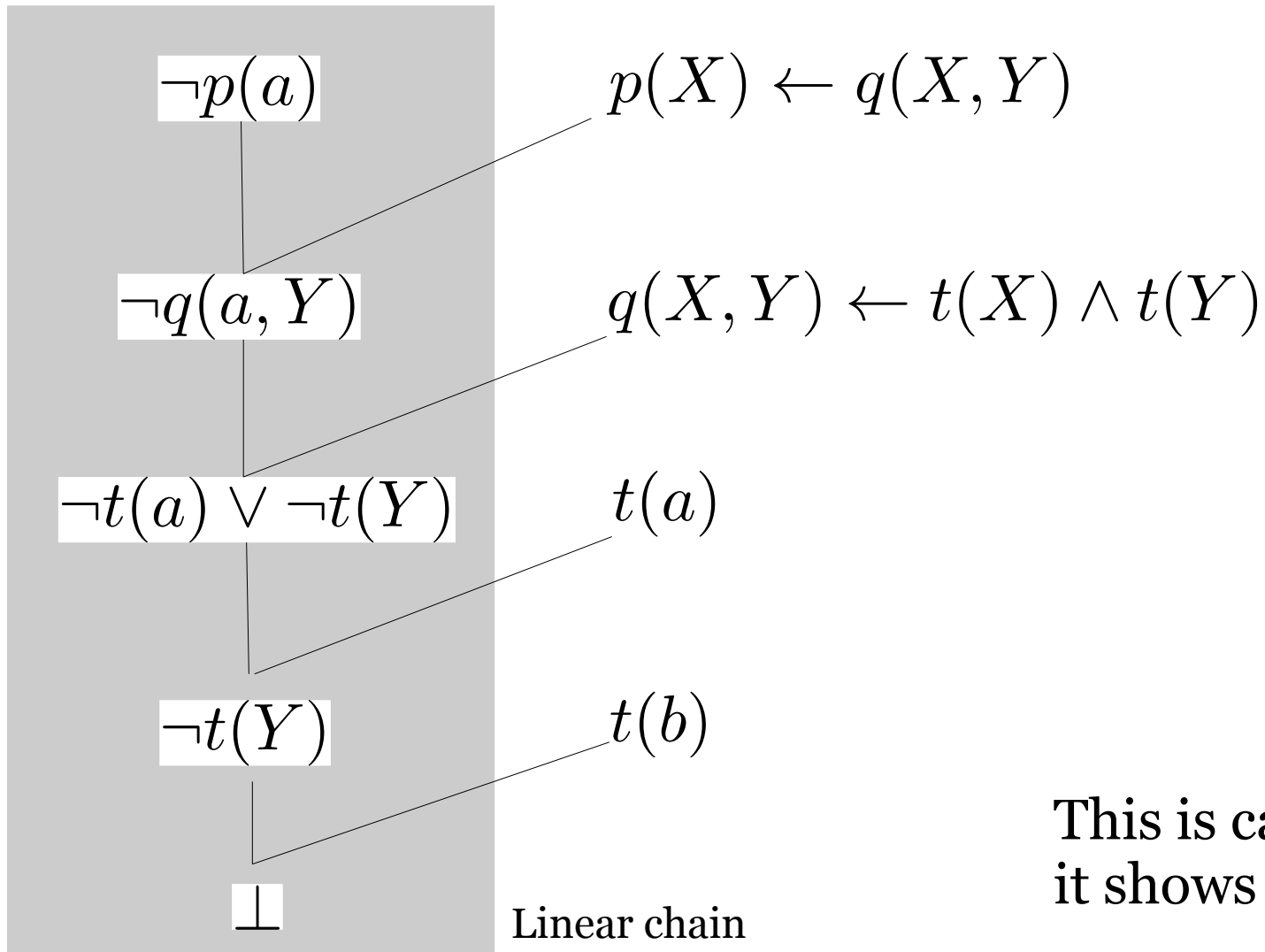
$p(a, X) \vee q(b), \neg q(Z) \vdash_R p(a, X)$

# SLD Resolution

- Selective Linear Definite clause resolution
- Resolution applied to a set with one *goal clause* and definite clauses otherwise, in which the goal clause is updated by means of resolution steps
- Resolution steps in this case form a linear chain
- Example:
  - Goal clause:  $\neg p(a)$
  - Definite clauses:  $p(X) \leftarrow q(X, Y)$   
 $q(X, Y) \leftarrow t(X) \wedge t(Y)$   
 $t(a)$   
 $t(b)$

Robert Kowalski

# SLD Resolution



$\neg p(a)$   
 $p(X) \leftarrow q(X, Y)$   
 $q(X, Y) \leftarrow t(X) \wedge t(Y)$   
 $t(a)$   
 $t(b)$

This is called an SLD proof for  $p(a)$ ;  
it shows that the definite clauses entail  $p(a)$



# SLD Resolution

- Search can be required to find a proof, or to determine that no proof exists

Goal:

$$\neg p(a)$$

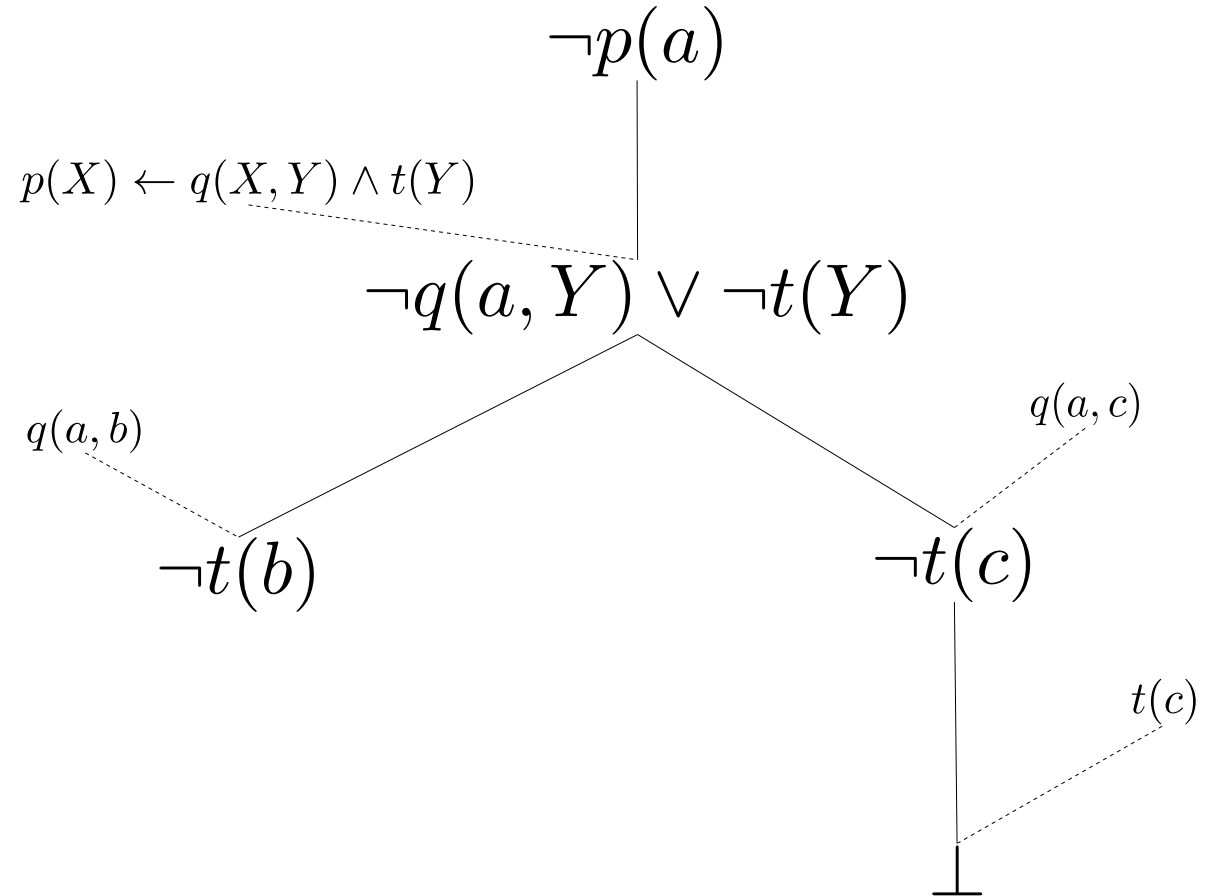
Definite clauses:

$$p(X) \leftarrow q(X, Y) \wedge t(Y)$$

$$q(a, b)$$

$$q(a, c)$$

$$t(c)$$

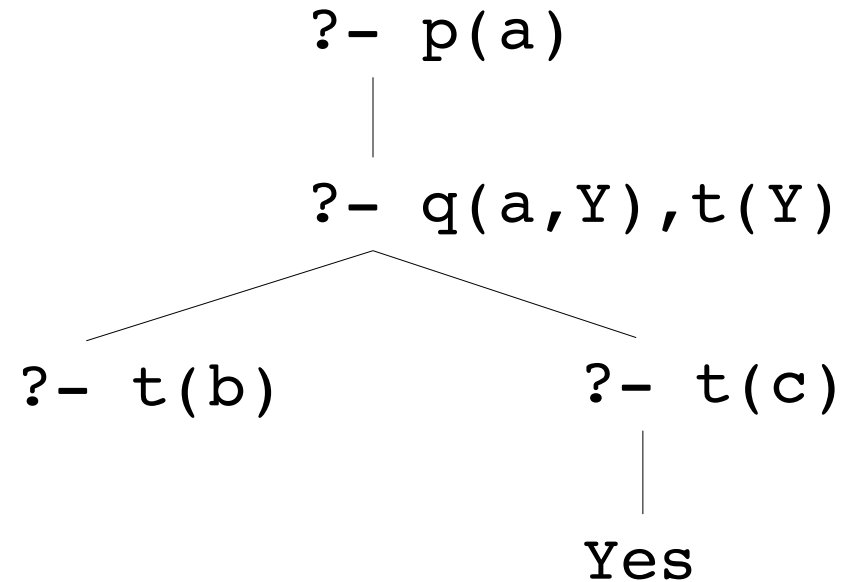


# Prolog

- Programming language based on SLD resolution
  - :- replaces  $\leftarrow$
  - , replaces  $\wedge$

```
p(X) :- q(X,Y), t(Y)
q(a,b).
q(a,c).
t(c).
```

```
?- p(a).
Yes
```



# Prolog

- Prolog applies resolution from top to bottom in the program
  - This can be important to avoid infinite recursion
  - Although it does not solve the recursion problem

```
p(X) :- p(f(X))  
p(f(a)).
```

```
?- p(a)
```

```
?- p(f(f(a)))
```

```
?- p(a)
```

```
    |  
?- p(f(a))
```

```
    |  
?- p(f(f(a)))
```

```
    |  
?- p(f(f(f(a))))
```

```
    |  
...
```

# Prolog

- Example: finding paths

```
path(X,Y) :- edge(X,Y).  
path(X,Y) :- edge(X,Z), path(Z,Y).
```

```
edge(a,b).  
edge(a,c).  
edge(c,d).
```

```
?- path(a,d).
```

```
?- path(X,Y).
```

← Generates all pairs of nodes between which a path exists

This is called the *extension* of the path predicate,  
i.e. all combinations of arguments for which the predicate is true.

# Prolog

- Support for functions: lists, mathematical operators

```
length([ ],0).
```

```
length([H|T],N) :-length(T,M), N is M+1.
```

```
member(X,[X|List]).
```

```
member(X,[Element|List]) :-member(X,List).
```

```
append([ ],List,List).
```

```
append([Element|L1],L2,[Element|L1L2]) :-append(L1,L2,L1L2).
```

```
?- length([a,b],X).
```

```
?- length([a,b],2).
```

```
?- member(b,[a,b,c]).
```

```
?- member(X,[a,b,c]).
```

# Prolog: Cuts

- Cuts are atoms that prevent backtracking in the search for proofs

`q(a,b).`

`q(a,c).`

`t(c).`

`p(X) :- q(X,Y), !, t(Y)`

`p(a).`

`?- p(a).`

No

- if the search reaches the cut, it will never consider alternatives for the current clause, or the variable assignments used at the moment in the clause
- the order of the program determines the order

- Cuts make Prolog less declarative...

# Prolog: Fail

- Fail is an atom that enforces that a proof fails

```
q(a,b).  
q(a,c).  
t(c).  
p(X) :- q(X,Y), fail.
```

```
?- p(a).  
No
```

- Combined with cuts, it can be used to express exceptions

```
like(X) :- chocolate(X), !, fail.  
like(X) :- sweet(X).  
sweet(fanta). sweet(mars). chocolate(mars).
```

```
?- like(mars).  
No
```

# Prolog: Negation

- Negation by failure: Prolog assumes that anything it can't prove, is false
- `not` is an atom defined as follows:

```
not(goal) :- call(goal), !, fail.  
not(goal).
```

- `not` can be used to check whether something can **not** be proved

```
like(X) :- sweet(X), not(chocolate(X)).  
sweet(fanta). sweet(mars). chocolate(mars).
```

```
?- like(mars).
```

No

```
?- like(fanta).
```

Yes



# Prolog: Negation

- Negation should be used **safely**: variables used in the negated atoms should either
  - Be bound
  - Not occur elsewhere in the clause

- Not safe:

```
p(X) :- not(q(X)).
```

- Safe:

```
p(X) :- q(X), not(t(X)).
```

```
p(X) :- q(X), not(t(X,Y)).
```

# Prolog: Graph Coloring

- Example code:

```
edge(a,b). edge(b,c). edge(a,c).  
edge(b,a). edge(c,b). edge(c,a).
```

```
sol(R,G,B) :- search(R,G,B,[a,b,c]).
```

```
no_conflict(C,[]).  
no_conflict(C,[C2|L]) :- not(edge(C,C2)), no_conflict(C,L).
```

```
search([],[],[],[]).  
search([C|R],G,B,[C|L]) :- search(R,G,B,L), no_conflict(C,R).  
search(R,[C|G],B,[C|L]) :- search(R,G,B,L), no_conflict(C,G).  
search(R,G,[C|B],[C|L]) :- search(R,G,B,L), no_conflict(C,B).
```

- Is this efficient?

# Exercise: Hamilton Path Problem

- Implement the Hamilton Path Problem in Prolog

```
edge(a,b). edge(b,c). edge(a,c).  
edge(b,a). edge(c,b). edge(c,a).
```

```
sol(L) :- search(L,[a,b,c]),no_conflict(L).
```

```
no_conflict([C]).  
no_conflict([C|[C2|L]]) :- edge(C,C2),no_conflict([C2|L]).
```

```
search([],[]).  
search([C|L],L2) :- member(C,L2),subtract(L2,[C],L3),search(L,L3).
```

# Seminar

- The seminar starts March 8 or March 15
- Everybody presents for 45 minutes (one part of a lecture)
- The topics will always be paired on one day, where both topics on the same day will be related
- In groups of 2 you will need to make an exercise related to the topic
- Every group delivers one report