Programming Languagages (Langages Evolués)

Roel Wuyts

Logic Programming

Overview

- Formal foundations
  - Propositional logic
  - First order predicate logic
- Logic programming: program specified declaratively
- Most widely used logic language: Prolog

Declarative Programming

- In imperative programming:
  Program = Data + Control
  - Specifies *how* the problem is solved
- In declarative programming:
  Program = Data
  - Specifies *what* problem to solve

Logic Programming View

- a computer program = a logical theory
- a procedure call = a theorem of which the truth needs to be established
- program execution = searching for a proof
- computation = automated logic deduction

More declarative programming

- Constraint solving
  - constrain possible values (domains) for variables
  - order of constraint is not important
  - constraint logic programming: allows constraints in logic programs
- Database query languages
  - e.g. SQL

Theoretical Foundations

- Predicate Logic
- Clausal form
- Horn clauses

Predicate Logic

negation	$\neg$	$\neg a$
conjunction	$\wedge$	$a \wedge b$
disjunction	$\vee$	$a \vee b$
equivalence	$\Leftrightarrow$	$a \Leftrightarrow b$
implication	$\Leftarrow$	$b \Leftarrow a$
existential quantifier	$\exists$	$\exists X.human(X)$
universal quantifier	$\forall$	$\forall X.human(X)$

$\exists X . (parent(mary,X) \wedge male(X))$

$\forall X . (human(X) \Leftarrow female(X))$

Clausal Forms

- Every proposition can be rewritten in clausal form:

  $$H1 \vee H2 \vee \ldots \vee Hn \Leftarrow A1 \wedge A2 \wedge \ldots \wedge An$$

  (if all A are true, then at least one H is true)

- By consequence:
  - existential quantifiers are not needed
  - universal quantifiers are implicit in the variables

Horn Clauses

- A Horn clause is a clausal form of form:

  $H \Leftarrow A1 \wedge A2 \wedge \ldots \wedge An$

  (H is true only if all A are true)

  - Note: different forms are possible:
    - H (query)
    - $H \Leftarrow$ true (fact)
    - $H \Leftarrow A1 \wedge A2 \wedge \ldots \wedge An$ (rule)

Principles of Logic Programming

- Resolution
- Unification
- Substitution
- Backtracking

Resolution and Substitution

- If a subgoal of a Horn clause unifies with the head of another Horn clause, resolution allows us to substitute that subgoal by the body of the unifying Horn clause.

- When we have two Horn clauses:

  $H \Leftarrow Terms$

  $T \Leftarrow T1 \wedge H \wedge T2$

- Then we can resolve the 2nd clause by *substitution*:

  $T \Leftarrow T1 \wedge Terms \wedge T2$

Unification

- Unification is the mechanism of substituting variables in logic terms with the goal of making them identical
  - It binds variables to corresponding values in a matching clause

Unification Examples

speaks(mary,english)
talkswith(X,Y) $\Leftarrow$ speaks(X,L), speaks(Y,L), X≠Y

- resolves in

  talkswith(mary,Y) $\Leftarrow$
  speaks(mary,english),speaks(Y,english),mary≠Y

- by substituting X=mary and L=english in the second clause

Unification Examples (ctd)

human(F) $\Leftarrow$ woman(F)
human(H) $\Leftarrow$ man(H)
eat(X) $\Leftarrow$ human(X), mouth(X,B)

- resolves in:
  - eat(X) $\Leftarrow$ woman(X), mouth(X,B)
    - substituting X=F in the first clause
  - eat(X) $\Leftarrow$ man(X), mouth(X,B)
    - substituting X=H in the second clause

Unification rules

- A constant unifies only with itself:

```
?- charles = charles.
  yes
?- charles = andrew.
  no
```

- An uninstantiated variable unifies with anything:

```
?- parent(charles, elizabeth) = Y.
 Y = parent(charles,elizabeth)
```

- A structured term unifies with another term only if it has the same function name and number of arguments, and the arguments can be unified recursively:

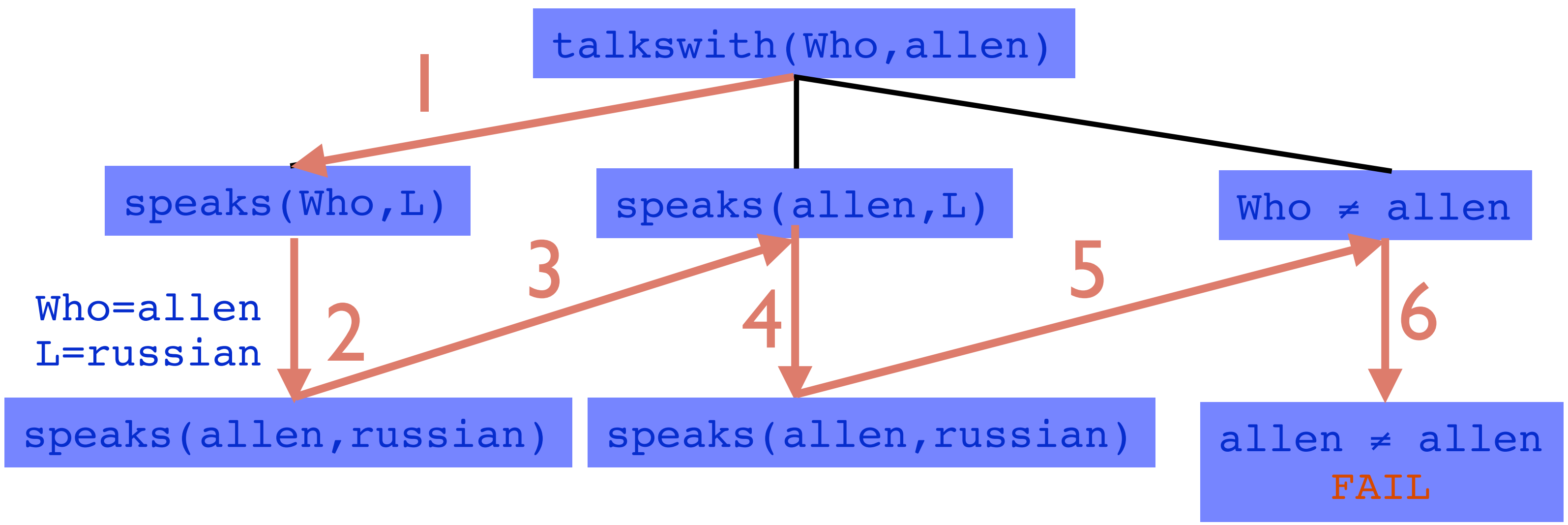
```
?- parent(charles, P) = parent(X, elizabeth).
 P = elizabeth,
 X = charles
```

Backtracking

- Search operation that, when a choice made in a search node lead to an inacceptable result, consists of returning to that node to make another choice
  - implicit control mechanism

Backtracking Example

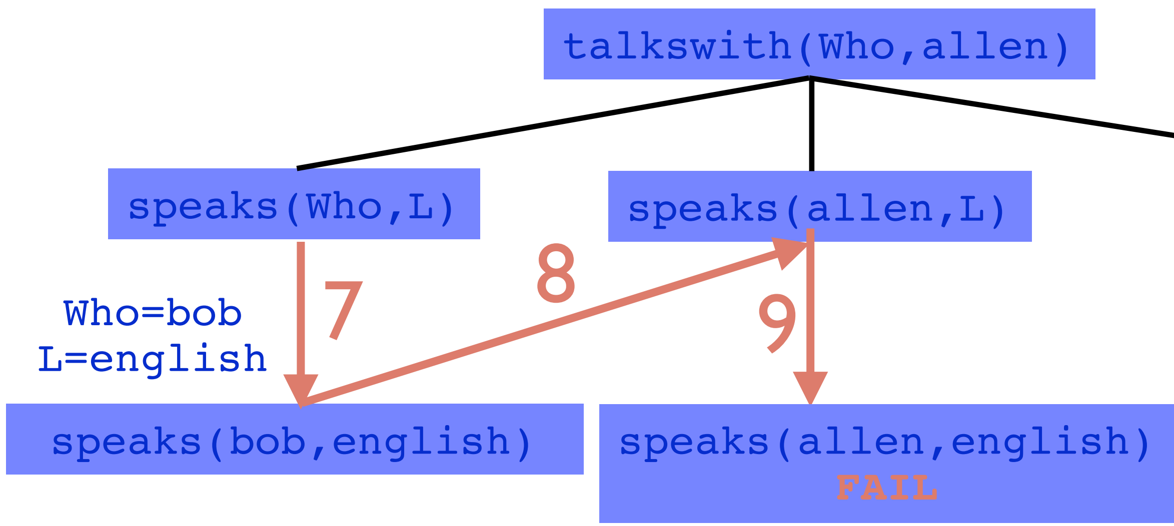
```
talkswith(X,Y) ⇐ speaks(X,L),speaks(Y,L),X≠Y
speaks(allen,russian)
speaks(bob,english)
speaks(mary,russion)
speaks(mary,english)
```



Go back to last substitution,
and propose next substitution

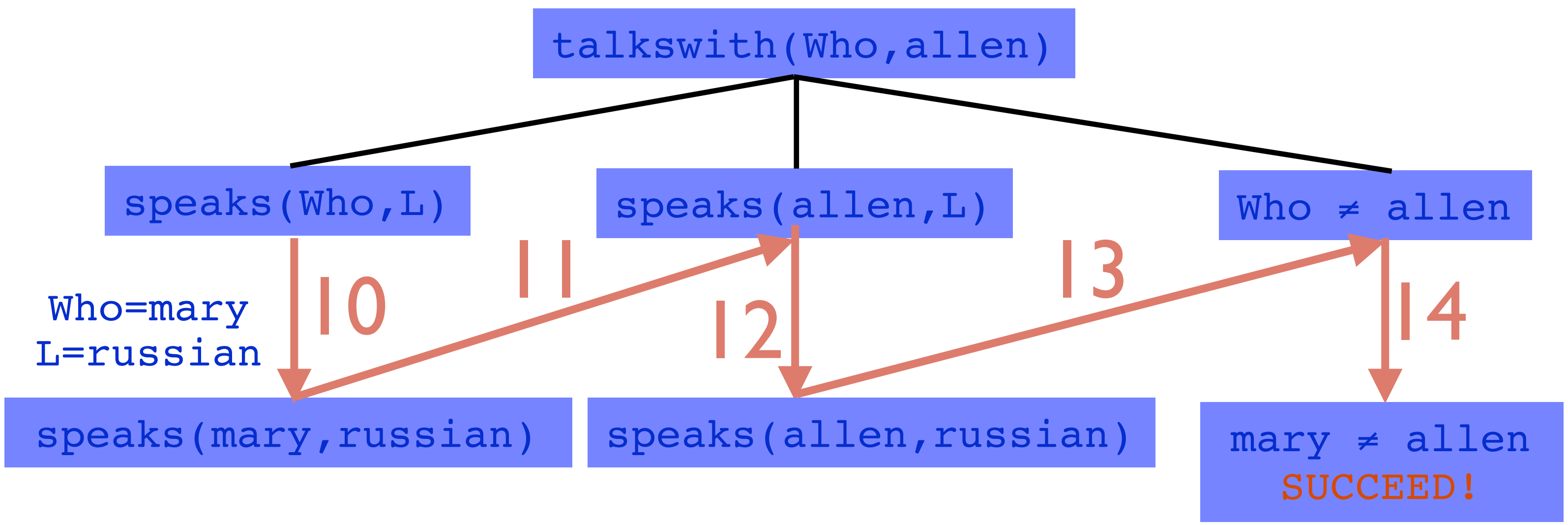
Backtracking Example (ctd)

```
talkswith(X,Y) ⇐ speaks(X,L),speaks(Y,L),X≠Y
speaks(allen,russian)
speaks(bob,english)
speaks(mary,russion)
speaks(mary,english)
```



Backtracking Example (ctd)

```
talkswith(X,Y) ⇐ speaks(X,L),speaks(Y,L),X≠Y
speaks(allen,russian)
speaks(bob,english)
speaks(mary,russion)
speaks(mary,english)
```



Multi-directionality

```
double(1,2)
double(2,4)
double(3,6)
```

```
? double(2,4)  --> succeeds

? double(3,X)  --> 6

? double(X,4)  --> 2

? double(X,Y)  --> (1,2) , (2,4) , (3,6)
```

Describes a *mathematical* relation !

Where bidirectionality ends...

```
square(N1,N2) :- N2 is N1*N1.

?- square(5,N2).
N2 = 25

?- square(5,25).
Yes

?- square(N1,25)
ERROR: Arguments are not sufficiently instantiated
```

- Solution: Constraint programming

Wrap-up

- Logic programming is *declarative*, not imperative
  - describe *what* the problem is, not *how* to solve it
  - relations are mathematical
- Based on first-order predicate logic, horn clauses
- Key foundations:
  - resolution, substitution, unification, backtracking

References

- http://www.ulb.ac.be/di/rwuyts/INFO020_2003/