

## Resolution Preliminaries

### Resolution Principle

The *Resolution Principle* is a rule of inference.

Using the Resolution Principle alone (without axiom schemata or other rules of inference), it is possible to build a theorem prover that is sound and complete for all of Relational Logic.

The search space using the Resolution Principle is much smaller than with standard axiom schemata.

## Plan

### First Lecture - Resolution Preliminaries

- Relational Clausal Form
- Unification

### Second Lecture - Resolution Principle

- Resolution Principle and Factoring
- Resolution Theorem Proving

### Third Lecture - Resolution Applications

- Theorem Proving
- Answer Extraction
- Reduction

### Fourth Lecture - Resolution Strategies

- Elimination Strategies (tautology elimination, subsumption, ...)
- Restriction Strategies (ancestry filtering, set of support, ...)

3

## Clausal Form

Relational resolution works only on expressions in *clausal form*.

Fortunately, it is possible to convert any set of Relational Logic sentences into an equally satisfiable set of sentences in clausal form.

4

## Clausal Form

A *literal* is either an atomic sentence or a negation of an atomic sentence.

A *clausal sentence* is either a literal or a disjunction of literals.

A *clause* is a set of literals (interpreted as disjunction).

$$\begin{aligned} &\{p(a)\} \\ &\{\neg p(a)\} \\ &\{p(x), q(x)\} \end{aligned}$$

The empty clause  $\{\}$  is unsatisfiable.

5

## Inseado

Implications Out:

$$\varphi_1 \Rightarrow \varphi_2 \quad \rightarrow \quad \neg \varphi_1 \vee \varphi_2$$

$$\varphi_1 \Leftarrow \varphi_2 \quad \rightarrow \quad \varphi_1 \vee \neg \varphi_2$$

$$\varphi_1 \Leftrightarrow \varphi_2 \quad \rightarrow \quad (\neg \varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg \varphi_2)$$

Negations In:

$$\neg \neg \varphi \quad \rightarrow \quad \varphi$$

$$\neg(\varphi_1 \wedge \varphi_2) \quad \rightarrow \quad \neg \varphi_1 \vee \neg \varphi_2$$

$$\neg(\varphi_1 \vee \varphi_2) \quad \rightarrow \quad \neg \varphi_1 \wedge \neg \varphi_2$$

$$\neg \forall v. \varphi \quad \rightarrow \quad \exists v. \neg \varphi$$

$$\neg \exists v. \varphi \quad \rightarrow \quad \forall v. \neg \varphi$$

6

## Inseado (continued)

Standardize variables

$$\forall x.p(x) \vee \forall x.q(x) \rightarrow \forall x.p(x) \vee \forall y.q(y)$$

Existentials Out (Outside in)

$$\exists x.p(x) \rightarrow p(a)$$

$$\forall x.(p(x) \wedge \exists z.q(x, y, z)) \rightarrow \forall x.(p(x) \wedge q(x, y, f(x, y)))$$

7

## Inseado (continued)

Alls Out

$$\forall x.(p(x) \wedge q(x, y, f(x, y))) \rightarrow p(x) \wedge q(x, y, f(x, y))$$

Distribution

$$\varphi_1 \vee (\varphi_2 \wedge \varphi_3) \rightarrow (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)$$

$$(\varphi_1 \wedge \varphi_2) \vee \varphi_3 \rightarrow (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3)$$

$$\varphi \vee (\varphi_1 \vee \dots \vee \varphi_n) \rightarrow (\varphi \vee \varphi_1 \vee \dots \vee \varphi_n)$$

$$(\varphi_1 \vee \dots \vee \varphi_n) \vee \varphi \rightarrow (\varphi_1 \vee \dots \vee \varphi_n \vee \varphi)$$

$$\varphi \wedge (\varphi_1 \wedge \dots \wedge \varphi_n) \rightarrow (\varphi \wedge \varphi_1 \wedge \dots \wedge \varphi_n)$$

$$(\varphi_1 \wedge \dots \wedge \varphi_n) \wedge \varphi \rightarrow (\varphi_1 \wedge \dots \wedge \varphi_n \wedge \varphi)$$

8

## Inseado (concluded)

### Operators Out

$$\begin{aligned} \varphi_1 \wedge \dots \wedge \varphi_n &\rightarrow \varphi_1 \\ &\dots \\ &\varphi_n \\ \varphi_1 \vee \dots \vee \varphi_n &\rightarrow \{\varphi_1, \dots, \varphi_n\} \end{aligned}$$

9

## Example

$$\begin{aligned} &\exists y.(g(y) \wedge \forall z.(r(z) \Rightarrow f(y,z))) \\ \text{I} &\exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y,z))) \\ \text{N} &\exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y,z))) \\ \text{S} &\exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y,z))) \\ \text{E} &g(\text{greg}) \wedge \forall z.(\neg r(z) \vee f(\text{greg}, z)) \\ \text{A} &g(\text{greg}) \wedge (\neg r(z) \vee f(\text{greg}, z)) \\ \text{D} &g(\text{greg}) \wedge (\neg r(z) \vee f(\text{greg}, z)) \\ \text{O} &\{g(\text{greg})\} \\ &\{\neg r(z), f(\text{greg}, z)\} \end{aligned}$$

10

## Example

- $\neg \exists y.(g(y) \wedge \forall z.(r(z) \Rightarrow f(y, z)))$   
I  $\neg \exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y, z)))$   
N  $\neg \exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y, z)))$   
 $\forall y.\neg(g(y) \wedge \forall z.(\neg r(z) \vee f(y, z)))$   
 $\forall y.(\neg g(y) \vee \neg \forall z.(\neg r(z) \vee f(y, z)))$   
 $\forall y.(\neg g(y) \vee \exists z.\neg(\neg r(z) \vee f(y, z)))$   
 $\forall y.(\neg g(y) \vee \exists z.(\neg \neg r(z) \wedge \neg f(y, z)))$   
 $\forall y.(\neg g(y) \vee \exists z.(r(z) \wedge \neg f(y, z)))$   
S  $\forall y.(\neg g(y) \vee \exists z.(r(z) \wedge \neg f(y, z)))$

11

## Example (concluded)

- $\forall y.(\neg g(y) \vee \exists z.(r(z) \wedge \neg f(y, z)))$   
E  $\forall y.(\neg g(y) \vee (r(h(y)) \wedge \neg f(y, h(y))))$   
A  $\neg g(y) \vee (r(h(y)) \wedge \neg f(y, h(y)))$   
D  $(\neg g(y) \vee r(h(y))) \wedge (\neg g(y) \vee \neg f(y, h(y)))$   
O  $\{\neg g(y) \vee r(h(y))\}$   
 $\{\neg g(y) \vee \neg f(y, h(y))\}$

12

## Clausal Form

**Bad News:** The result of converting a set of sentences is not necessarily logically equivalent to the original set of sentences. Why? Introduction of Skolem constants and functions.

**Good News:** The result of converting a set of sentences is satisfiable if and only if the original set of sentences is satisfiable. Important because we use satisfiability to determine logical entailment.

13

## Difficulty with Universal Instantiation

$$\forall x.p(x,b)$$

$$\forall y.\neg p(a,y)$$

$$p(a,b)$$

$$\neg p(a,b)$$

14

## Substitutions

A *substitution* is a finite set of pairs of variables and terms, called *replacements*.

$$\{X \leftarrow a, Y \leftarrow f(b), V \leftarrow W\}$$

The result of applying a substitution  $\sigma$  to an expression  $\varphi$  is the expression  $\varphi\sigma$  obtained from  $\varphi$  by replacing every occurrence of every variable in the substitution by its replacement.

$$p(X, X, Y, Z) \{X \leftarrow a, Y \leftarrow f(b), V \leftarrow W\} = p(a, a, f(b), Z)$$

15

## Cascaded Substitutions

$$r\{x, y, z\} \{x \leftarrow a, y \leftarrow f(u), z \leftarrow v\} = r\{a, f(u), v\}$$

$$r\{a, f(u), v\} \{u \leftarrow d, v \leftarrow e\} = r\{a, f(d), e\}$$

$$r\{x, y, z\} \{x \leftarrow a, y \leftarrow f(d), z \leftarrow e\} = r\{a, f(d), e\}$$

16

## Composition of Substitutions

The *composition* of substitution  $\sigma$  and  $\tau$  is the substitution (written  $compose(\sigma, \tau)$  or, more simply,  $\sigma\tau$ ) obtained by

- (1) applying  $\tau$  to the replacements in  $\sigma$
- (2) adding to  $\sigma$  pairs from  $\tau$  with different variables
- (3) deleting any assignments of a variable to itself.

$$\begin{aligned} & \{x \leftarrow a, y \leftarrow f(u), z \leftarrow v\} \{u \leftarrow d, v \leftarrow e, z \leftarrow g\} \\ &= \{x \leftarrow a, y \leftarrow f(d), z \leftarrow e\} \{u \leftarrow d, v \leftarrow e, z \leftarrow g\} \\ &= \{x \leftarrow a, y \leftarrow f(d), z \leftarrow e, u \leftarrow d, v \leftarrow e\} \end{aligned}$$

17

## Unification

A substitution  $\sigma$  is a *unifier* for an expression  $\varphi$  and an expression  $\psi$  if and only if  $\varphi\sigma = \psi\sigma$ .

$$\begin{aligned} p(X, Y) \{X \leftarrow a, Y \leftarrow b, V \leftarrow b\} &= p(a, b) \\ p(a, V) \{X \leftarrow a, Y \leftarrow b, V \leftarrow b\} &= p(a, b) \end{aligned}$$

If two expressions have a unifier, they are said to be *unifiable*. Otherwise, they are *nonunifiable*.

$$\begin{aligned} & p(X, X) \\ & p(a, b) \end{aligned}$$

18

## Non-Uniqueness of Unification

Unifier 1:

$$\begin{aligned}p(X, Y) \{X \leftarrow a, Y \leftarrow b, V \leftarrow b\} &= p(a, b) \\ p(a, V) \{X \leftarrow a, Y \leftarrow b, V \leftarrow b\} &= p(a, b)\end{aligned}$$

Unifier 2:

$$\begin{aligned}p(X, Y) \{X \leftarrow a, Y \leftarrow f(W), V \leftarrow f(W)\} &= p(a, f(W)) \\ p(a, V) \{X \leftarrow a, Y \leftarrow f(W), V \leftarrow f(W)\} &= p(a, f(W))\end{aligned}$$

Unifier 3:

$$\begin{aligned}p(X, Y) \{X \leftarrow a, Y \leftarrow V\} &= p(a, V) \\ p(a, V) \{X \leftarrow a, Y \leftarrow V\} &= p(a, V)\end{aligned}$$

19

## Most General Unifier

A substitution  $\sigma$  is a *most general unifier (mgu)* of two expressions if and only if it is as general as or more general than any other unifier.

**Theorem:** If two expressions are unifiable, then they have an mgu that is unique up to variable permutation.

$$\begin{aligned}p(X, Y) \{X \leftarrow a, Y \leftarrow V\} &= p(a, V) \\ p(a, V) \{X \leftarrow a, Y \leftarrow V\} &= p(a, V)\end{aligned}$$

$$\begin{aligned}p(X, Y) \{X \leftarrow a, V \leftarrow Y\} &= p(a, Y) \\ p(a, V) \{X \leftarrow a, V \leftarrow Y\} &= p(a, Y)\end{aligned}$$

20

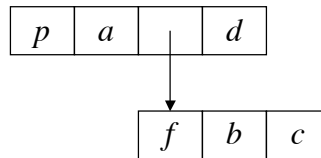
## Expression Structure

Each expression is treated as a sequence of its immediate subexpressions.

Linear Version:

$$p(a, f(b, c), d)$$

Structured Version:



21

## Most General Unification (preliminary)

```
function mgu (x, y, s)  
  { if (varp(x)) {return mguvar(x, y, s)};  
  if (atom(x)) {return mguatom(x, y, s)};  
  if (varp(y)) {return mguvar(y, x, s)};  
  if (atom(y)) {return false};  
  for (var i=0; i < x.length; i++)  
    { s = mgu(x[i], y[i], s);  
    if (s == false) {return false}};  
  return s}
```

22

## Most General Unification (preliminary)

```
function mguatom (x, y, s)  
{if (varp(y)) {return mguvar(y, x, s)};  
  if (x == y) {return s};  
  return false}
```

```
function mguvar (x, y, s)  
{if (x == y) {return s};  
  var dum = getbinding(x, s);  
  if (dum != false) {return mgu(dum, y, s)};  
  return compose(s, {x ← plug(y, s)})}
```

23

## Example

**Call:**  $mgu(p(X, b), p(a, Y), \{\})$

**Call:**  $mgu(p, p, \{\})$

**Exit:**  $\{\}$

**Call:**  $mgu(X, a, \{\})$

**Exit:**  $\{X \leftarrow a\}$

**Call:**  $mgu(b, Y, \{X \leftarrow a\})$

**Exit:**  $\{X \leftarrow a, Y \leftarrow b\}$

**Exit:**  $\{X \leftarrow a, Y \leftarrow b\}$

24

## Example

**Call:**  $mgu(p(X, X), p(a, b), \{\})$

**Call:**  $mgu(p, p, \{\})$

**Exit:**  $\{\}$

**Call:**  $mgu(X, a, \{\})$

**Exit:**  $\{X \leftarrow a\}$

**Call:**  $mgu(X, b, \{X \leftarrow a\})$

**Call:**  $mgu(a, b, \{X \leftarrow a\})$

**Exit:** false

**Exit:** false

**Exit:** false

25

## Example

**Call:**  $mgu(p(f(X), f(X)), p(Y, f(a)), \{\})$

**Call:**  $mgu(p, p, \{\})$

**Exit:**  $\{\}$

**Call:**  $mgu(f(X), Y, \{\})$

**Exit:**  $\{Y \leftarrow f(X)\}$

**Call:**  $mgu(f(X), f(a), \{Y \leftarrow f(X)\})$

**Call:**  $mgu(f, f, \{Y \leftarrow f(X)\})$

**Exit:**  $\{Y \leftarrow f(X)\}$

**Call:**  $mgu(X, a, \{Y \leftarrow f(X)\})$

**Exit:**  $\{Y \leftarrow f(a), X \leftarrow a\}$

**Exit:**  $\{Y \leftarrow f(a), X \leftarrow a\}$

**Exit:**  $\{Y \leftarrow f(a), X \leftarrow a\}$

26

## Example

**Call:**  $mgu(p(X, X), p(Y, f(Y)), \{\})$

**Call:**  $mgu(p, p, \{\})$

**Exit:**  $\{\}$

**Call:**  $mgu(X, Y, \{\})$

**Exit:**  $\{X \leftarrow Y\}$

**Call:**  $mgu(X, f(Y), \{X \leftarrow Y\})$

**Call:**  $mgu(Y, f(Y), \{X \leftarrow Y\})$

**Exit:**  $\{X \leftarrow f(Y), Y \leftarrow f(Y)\}$

**Exit:**  $\{X \leftarrow f(Y), Y \leftarrow f(Y)\}$

**Exit:**  $\{X \leftarrow f(Y), Y \leftarrow f(Y)\}$

27

## Problem

Circularity Problem:

$\{X \leftarrow f(Y), Y \leftarrow f(Y)\}$

Unification Problem:

$p(X, X) \{X \leftarrow f(Y), Y \leftarrow f(Y)\} = p(f(Y), f(Y))$

$p(Y, f(Y)) \{X \leftarrow f(Y), Y \leftarrow f(Y)\} = p(f(Y), f(f(Y)))$

Semantic Problem:

$\sim \text{hates}(X, X)$

$\text{hates}(Y, f(Y))$

28

## Solution

Before assigning a variable to an expression, first check that the variable does not occur within that expression.

This is called, oddly enough, the *occur check* test.

Prolog does not do the occur check (and is proud of it).

29

## Most General Unification (revised)

```
function mguvar (x, y, s)
  {if (x == y) {return s};
  var dum = getbinding(x, s);
  if (dum != false) {return mgu(dum, y, s)};
  if (mguchkp(x, y, s)) {return false};
  return compose(s, {x ← plug(y, s)})}

function mguchkp (x, y, s)
  {if (x == y) {return true};
  if (varp(y)) {return mguchkp(x, right(assoc(y, s)), s)};
  if (atom(y)) {return false};
  for (var i=0; i < y.length; i++)
    {if (mguchkp(x, y[i], s)) {return true}};
  return false}
```

30

## Example

**Call:**  $mgu(p(X, X), p(Y, f(Y)), \{\})$

**Call:**  $mgu(p, p, \{\})$

**Exit:**  $\{\}$

**Call:**  $mgu(X, Y, \{\})$

**Exit:**  $\{X \leftarrow Y\}$

**Call:**  $mgu(X, f(Y), \{X \leftarrow Y\})$

**Call:**  $mgu(Y, f(Y), \{X \leftarrow Y\})$

**Exit:** false

**Exit:** false

**Exit:** false

31

## Propositional Resolution

$$\frac{\begin{array}{l} \{\varphi_1, \dots, \varphi, \dots, \varphi_m\} \\ \{\psi_1, \dots, \neg \varphi, \dots, \psi_n\} \end{array}}{\{\varphi_1, \dots, \varphi_m, \psi_1, \dots, \psi_n\}}$$

32

## Relational Resolution (sort of)

$$\frac{\{\varphi_1, \dots, \varphi_m\} \quad \{\psi_1, \dots, \neg \psi, \dots, \psi_n\}}{\{\varphi_1, \dots, \varphi_m, \psi_1, \dots, \psi_n\} \sigma}$$

where  $\sigma = mgu(\varphi, \psi)$