

# Automated Reasoning

- Linear Resolution
- Linear Resolution with Negation
- Unification

# Substitutions

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

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

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.

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

# Unification

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

$$p(X, Y) \{X \mapsto a, Y \mapsto b, V \mapsto b\} = p(a, b)$$

$$p(a, V) \{X \mapsto a, Y \mapsto b, V \mapsto b\} = p(a, b)$$

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

$$p(X, X)$$

$$p(a, b)$$

# Non-Uniqueness of Unification

Unifier 1:

$$p(X, Y) \{X \mapsto a, Y \mapsto b, V \mapsto b\} = p(a, b)$$

$$p(a, V) \{X \mapsto a, Y \mapsto b, V \mapsto b\} = p(a, b)$$

Unifier 2:

$$p(X, Y) \{X \mapsto a, Y \mapsto f(W), V \mapsto f(W)\} = p(a, f(W))$$

$$p(a, V) \{X \mapsto a, Y \mapsto f(W), V \mapsto f(W)\} = p(a, f(W))$$

Unifier 3:

$$p(X, Y) \{X \mapsto a, Y \mapsto V\} = p(a, V)$$

$$p(a, V) \{X \mapsto a, Y \mapsto V\} = p(a, V)$$

# Most General Unifier

A substitution  $\sigma$  is *more general* than a substitution  $\theta$  if and only if there is a substitution  $\tau$  such that  $\sigma \circ \tau = \theta$ .

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

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

$$p(X, Y) \{X \mapsto a, Y \mapsto V\} = p(a, V)$$

$$p(a, V) \{X \mapsto a, Y \mapsto V\} = p(a, V)$$

$$p(X, Y) \{X \mapsto a, V \mapsto Y\} = p(a, Y)$$

$$p(a, V) \{X \mapsto a, V \mapsto Y\} = p(a, Y)$$

# Unification (1)

Terms encoded as lists:  $f(g(x),c) \rightsquigarrow [f,[g,x],c]$

```
function mgu(x,y,σ)  
begin  
  if  $x=y$  then return  $\sigma$ ;  
  if is_variable(x) then return mguvar(x,y,σ);  
  if is_variable(y) then return mguvar(y,x,σ);  
  if not is_list(x) then  
    if is_list(y) or  $x \neq y$  then return fail  
    else return  $\sigma$ ;  
  if not is_list(y) then return fail;  
   $\sigma := mgu(head(x),head(y),\sigma)$ ;  
  if  $\sigma=fail$  then return fail  
  else return mgu(tail(x),tail(y), $\sigma$ )  
end
```

## Unification (2 – Straightforward but Incorrect)

```
function mguvar(x,y, $\sigma$ )  
begin  
  if  $x \rightsquigarrow t \in \sigma$  then return mgu(t,y, $\sigma$ );  
  return  $\{x \rightsquigarrow y\} \cup \sigma$   
end
```

# Problem: The Occur Check

`same_person(X,X)`

`=> same_person(Y,father_of(Y))`

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

## Unification (2 – Corrected Version)

```
function mguvar(x,y, $\sigma$ )  
begin  
  if  $x \rightsquigarrow t \in \sigma$  then return mgu(t,y, $\sigma$ );  
  if mgucheck(x,y, $\sigma$ ) then return fail;  
  return  $\{x \rightsquigarrow y\} \cup \sigma$   
end
```

## Unification (3)

```
function mgucheck(x,y, $\sigma$ )  
begin  
  if  $x=y$  then return true;  
  if is_variable(y) then  
    if  $y \mapsto t \in \sigma$  then return mgucheck(x,t, $\sigma$ )  
    else return false;  
  if not is_list(y) or  $y=[]$  then return false;  
  if mgucheck(x,head(y), $\sigma$ ) then return true;  
  else return mgucheck(x,tail(y), $\sigma$ )  
end
```

# Example

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

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

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

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

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

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

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

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

# Example

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

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

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

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

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

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

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

**Exit:** fail

**Exit:** fail

**Exit:** fail

# Example

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

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

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

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

**Exit:**  $\{Y \mapsto [f, X]\}$

**Call:**  $mgu([f, X], [f, a], \{Y \mapsto [f, X]\})$

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

**Exit:**  $\{Y \mapsto [f, X]\}$

**Call:**  $mgu(X, a, \{Y \mapsto [f, X]\})$

**Exit:**  $\{X \mapsto a, Y \mapsto [f, X]\}$

**Exit:**  $\{X \mapsto a, Y \mapsto [f, X]\}$

**Exit:**  $\{X \mapsto a, Y \mapsto [f, X]\}$

# Example

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

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

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

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

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

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

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

**Call:**  $mgucheck(Y, [f, Y], \{X \mapsto Y\})$

**Exit:** true

**Exit:** fail

**Exit:** fail

**Exit:** fail

# Linear Resolution Step

Given:

Query  $L_1 \wedge L_2 \wedge \dots \wedge L_m$  (without negation)

Clauses  $\ell$  (without negation)

Let:

$A \Leftarrow B_1 \wedge \dots \wedge B_n$  “fresh” variant of a clause in  $\ell$

$\sigma$  mgu of  $L_1$  and  $A$

Then  $L_1 \wedge L_2 \wedge \dots \wedge L_m \rightarrow (B_1 \wedge \dots \wedge B_n \wedge L_2 \wedge \dots \wedge L_m)\sigma$

is a **linear resolution step**.

# Derivation

A sequence of resolution steps is called a **derivation**.

A **successful** derivation ends with the empty query.

The **answer substitution** (inferred by a successful derivation) is obtained by composing the mgu's  $\sigma_1 \circ \dots \circ \sigma_n$  of each step (and restricting the result to the variables in the original query).

A **failed** derivation ends with a query to which no clause applies.

# Linear Resolution with Negation

Given:

Query  $L_1 \wedge L_2 \wedge \dots \wedge L_m$

Clauses  $\ell$

- If  $L_1$  is an atom, proceed as before.
- If  $L_1$  is of the form  $\sim A$ , then
  - if all derivations for  $A$  **fail** then
$$L_1 \wedge L_2 \wedge \dots \wedge L_m \rightarrow L_2 \wedge \dots \wedge L_m$$
  - if there is a **successful** derivation for  $A$  then
$$L_1 \wedge L_2 \wedge \dots \wedge L_m \rightarrow \text{fail}$$

# Disjunction

A clause with a disjunction

$$A \leq B \wedge (C_1 \mid C_2) \wedge D$$

is logically equivalent to the conjunction of the clauses

$$A \leq B \wedge C_1 \wedge D$$

$$A \leq B \wedge C_2 \wedge D$$

# Example (1)

Clauses:

```
true(cell(1,1,b))
```

```
⋮
```

```
true(cell(3,3,b))
```

```
true(control(white))
```

```
legal(P,mark(X,Y)) <= true(cell(X,Y,b)) ∧ true(control(P))
```

```
legal(white,noop) <= true(cell(X,Y,b)) ∧ true(control(black))
```

```
legal(black,noop) <= true(cell(X,Y,b)) ∧ true(control(white))
```

Query `legal(P,M)` has answer substitutions

```
{P ↦ white, M ↦ mark(1,1)}, ..., {P ↦ white, M ↦ mark(3,3)}
```

```
{P ↦ black, M ↦ noop}
```

## Example (2)

Clauses:

`true(p)`

`does(robot,b)`

`next(p) <= does(robot,a) ∧ ~true(p)`  
`| does(robot,b) ∧ true(q)`

`next(q) <= does(robot,a) ∧ true(q)`  
`| does(robot,b) ∧ true(p)`

Query `next(Fluent)` has the only answer substitution

`{Fluent ↦ q}`

## Example (3)

Clauses:

```
player(red)
```

```
player(blue)
```

```
player(green)
```

```
true(freecell(blue))
```

```
trapped(P) <= ~true(freecell(P))
```

```
goal(P,100) <= player(P) ^ ~trapped(P)
```

Query `goal(P,100)` has the only computed answer

```
{P ↦ blue}
```

# Computing Answers

**function** *prove* (*query*,  $\ell$ )

*/\* return true if query has a successful derivation wrt.  $\ell$   
else return false \*/*

**function** *findone* (*x*, *query*,  $\ell$ )

*/\* return computed answer for expression x of a successful derivation for query wrt.  $\ell$  \*/*

**function** *findall* (*x*, *query*,  $\ell$ )

*/\* return the list of all answers for expression x for query wrt.  $\ell$  \*/*

# Example

$\ell$ :

```
true(cell(1,1,b))
```

```
⋮
```

```
true(cell(3,3,b))
```

```
true(control(white))
```

```
legal(P,mark(X,Y)) <= true(cell(X,Y,b)) ∧ true(control(P))
```

```
legal(white,noop) <= true(cell(X,Y,b)) ∧ true(control(black))
```

```
legal(black,noop) <= true(cell(X,Y,b)) ∧ true(control(white))
```

*findall*(M,legal(white,M), $\ell$ ) returns

```
[mark(1,1), ..., mark(3,3)]
```