L I P 6 C N R S

Automatic Theorem Proving

Tableau Method

Resolution in Propositional Logic

Unification

Resolution in First Order Logic

ACASA – Faculté des sciences de Sorbonne Université

L I P 6 C N R S

Automatic Theorem Proving

Tableau Method

Resolution in Propositional Logic

Unification

Resolution in First Order Logic

Resolution Rule in Propositional Logic

- Been given two clauses C_1 and C_2
- Been given an atomic proposition such that $A \in C_1$ and $\neg A \in C_2$
- The resolution of C_1 and C_2 by A and $\neg A$ is:
  $C = res(C_1, C_2; A, \neg A)$
  $= [C_1 - \{A\}] \vee [C_2 - \{\neg A\}]$

Example: if $C_1 = \neg man \vee mortal$,
$C_2 = \neg socrate \vee man$, $C_3 = socrate$ and
$C_4 = \neg mortal$

$C = res(C_3, C_2; socrate, \neg socrate) = man$

$C' = res(C_1, C_4 ; mortal, \neg mortal) = \neg man$

Generality of the Resolution

To prove $S \models C$, it is sufficient to prove that $S \cup \{\neg C\}$ is contradictory, i.e. that

$S \cup \{\neg C\} \models \square$

$\square$ denotes the empty clause, i.e. the false

Theorem: $S \models C$ if and only if it is possible to derive the empty clause by iterative application of the resolution rule on $S \cup \{\neg C\}$

L I P 6 C N R S

Automatic Theorem Proving

Tableau Method

Resolution in Propositional Logic

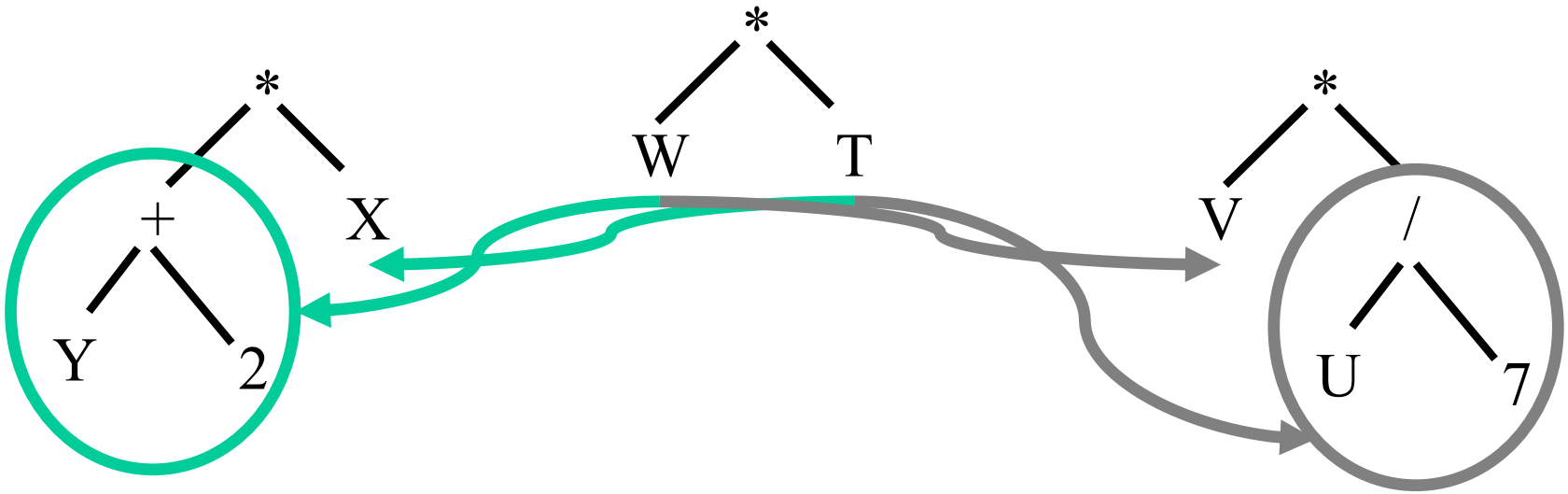
Unification

Resolution in First Order Logic

Pattern matching

Is a term an instance of another?

Pattern matching: the term t_1 match with the term t_2 if and only if there exists a substitution σ such that: $\mathbf{t_1\sigma = t_2}$



Representing a term with a list

- **S-expression**:
  - An *S-expression* is either an *atom (character string)*
  - Or a *sequence* (possibly empty) of S-expression
- *Examples*:
  - []
  - ‘a’, ‘Salut’, ‘Insoluble’, ‘Rien!’, 421
  - [‘a’, [‘Salut’, ‘à’, ‘toi’], 421, [‘Rien’, [‘vraiment rien!’]]]
- **Term**:
  - **Definition**: set of functions $\mathcal{F}$, set of variables $\mathcal{V}$
  - **Representation with a S-expression**: (3 * (?x + (?y - ?x)))
    [‘*’, 3, [‘+’, ‘?x’, [‘-’, ‘?y’, ‘?x’]]]

Pattern matching algorithm

filtering(t1, t2)

If t1 = t2 **then** σ := {} **return;**

If t1 = () or t2 = () **then** σ := fail; **return;**

If var(t1) **then** σ := {{t1 / t2}}; **return;**

If atom(t1) or atom(t2) **then** σ := fail; **return;**

σ_1 := filtering(head (t1) , head(t2));

If σ_1 = fail **then** σ := fail ; **return;**

else σ_2 := filtering(substitute(σ_1 , tail(t1)) , tail(t2));

If σ_2 = fail **then** σ := fail; **return;**

else σ := composition (σ_1 , σ_2)

Filtering in "Python"

```
def filtrage_(t1, t2, v1):
    if t1 == t2:
        return { }
    if var(t1):
        if t1 in v1:
            sigma = {}
            sigma[t1] = t2
        else: sigma = 'echec'
        return sigma
    if ( t1 == [] or t2 == []):
        return 'echec'
    if ( type(t1) <> type([]) or  type(t2) <> type([])):
        return 'echec'
    sigma1 = filtrage_(t1[0], t2[0], v1)
    if (sigma1 <> 'echec' and len(t1) > 1 and  len(t2) > 1 ):
        sigma2 = filtrage_(appliquer_substitution(t1[1:], sigma1), t2[1:], v1)
        return composer_substitution(sigma2, sigma1)
    elif t1[1:] <> t2[1:]:
        return 'echec'
    else: return sigma1
```

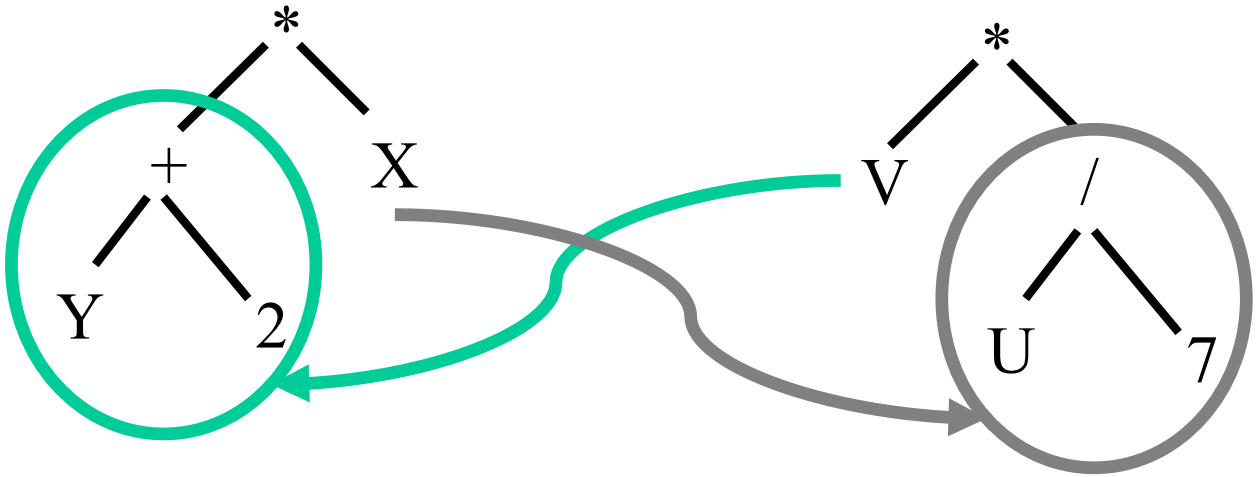
```
def filtrage(t1, t2):
    vars1 = variables(t1)
    return filtrage_(t1, t2, vars1)
```

Unification

Unification: the terms t_1 and t_2 unify if and only if there exists a substitution σ such that:

$t_1\sigma = t_2\sigma$

Pattern matching ⇒ Unification

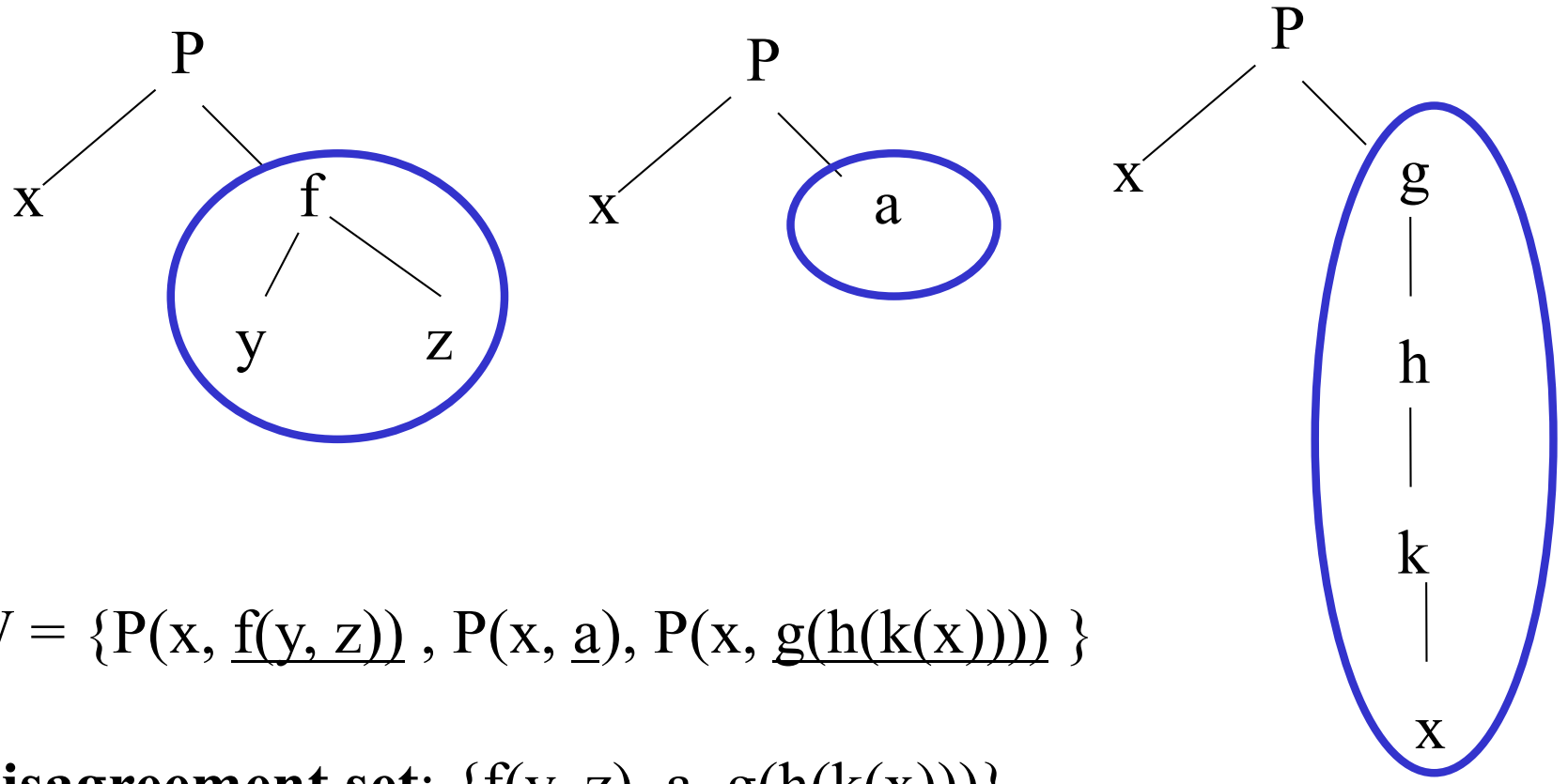


Unification Algorithm (*preliminaries*)

Definition: *a disagreement set* of a set of terms W is obtained by detecting the first difference in non identical terms (depth first search).

Example: W = {P(x, f(y, z)) , P(x, a), P(x, g(h(k(x)))) }

Disagreement set: *example*



W = {P(x, f(y, z)) , P(x, a), P(x, g(h(k(x)))) }

Disagreement set: {f(y, z), a, g(h(k(x)))}

Unification algorithm for a set of terms W

Step 1: $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$

Step 2: if W_k is a singleton then stop; σ_k is the MGU (*most general unifier)* of W.
else, find the disagreement set D_k of W_k

Step 3: if there exists a variable v_k and a term t_k belonging to D_k such that v_k does not appear in t_k, then go to step 4.
else, stop; W is not unifiable

Step 4: $\sigma_{k+1} := \sigma_k \{v_k / t_k\}$; $W_{k+1} := W_k \{v_k / t_k\}$ *[note that* $W_{k+1} = W\sigma_{k+1}$ *]*

Step 5: $k := k+1$; go to step 2

Unification algorithm: *example*

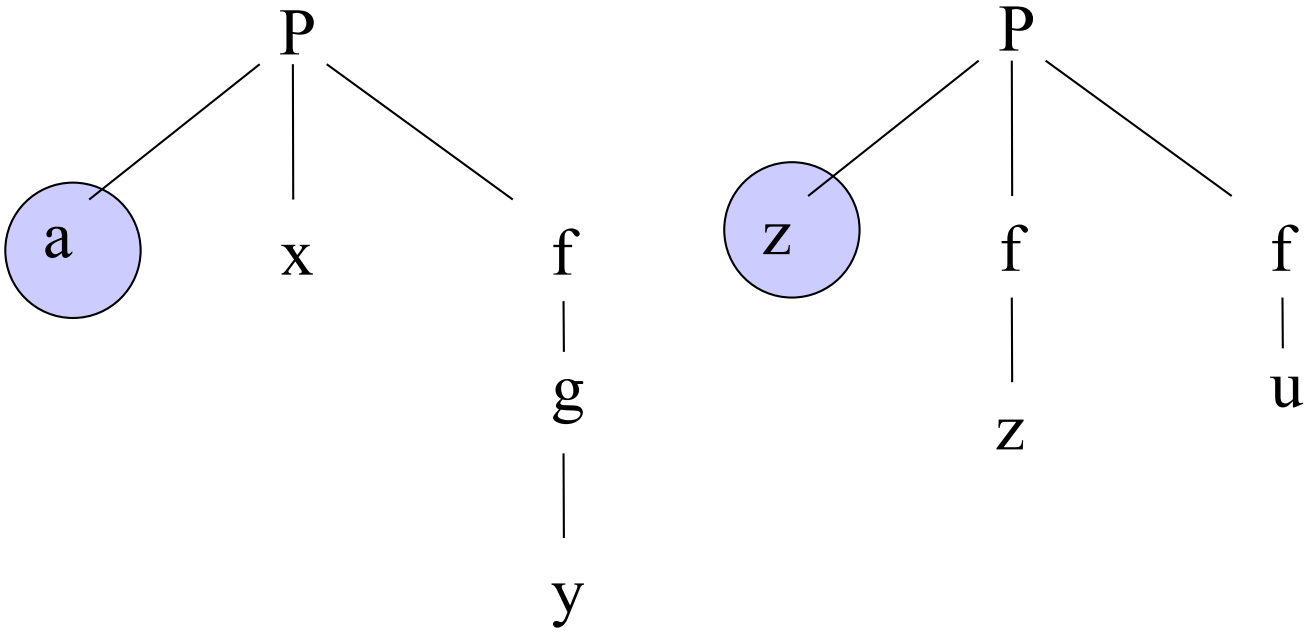
$W = \{ P(a, x, f(g(y))), P(z, f(z), f(u))\}$

$k = 0$

$W_k = W,$

$\sigma_k = \varepsilon$

$D_k = \{a, z\}$



Unification algorithm: *example*

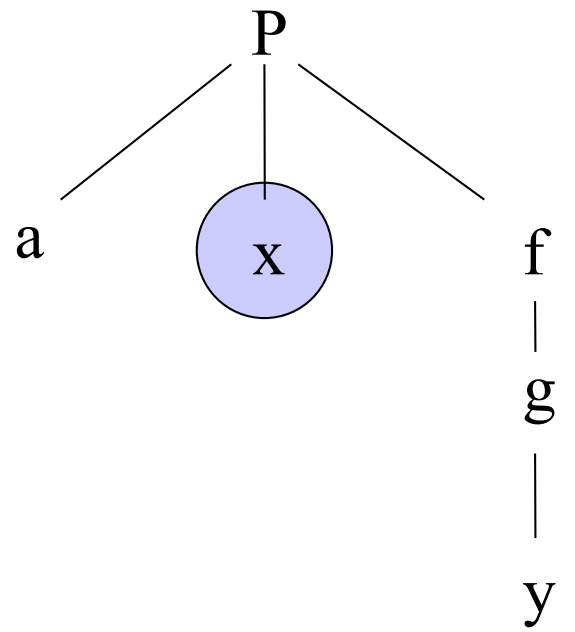
$W = \{ P(a, x, f(g(y))), P(z, f(z), f(u))\}$

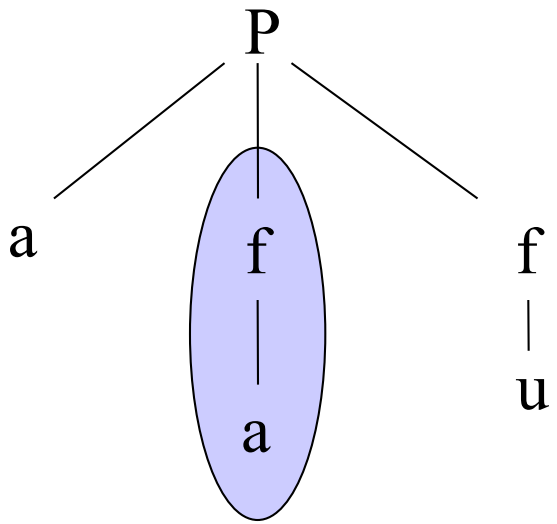
$k = 1$

$W_k = \{ P(a, x, f(g(y))), P(a, f(a), f(u))\}$

$\sigma_k = \{z / a\}$

$D_k = \{x, f(a)\}$





Unification algorithm: *example*

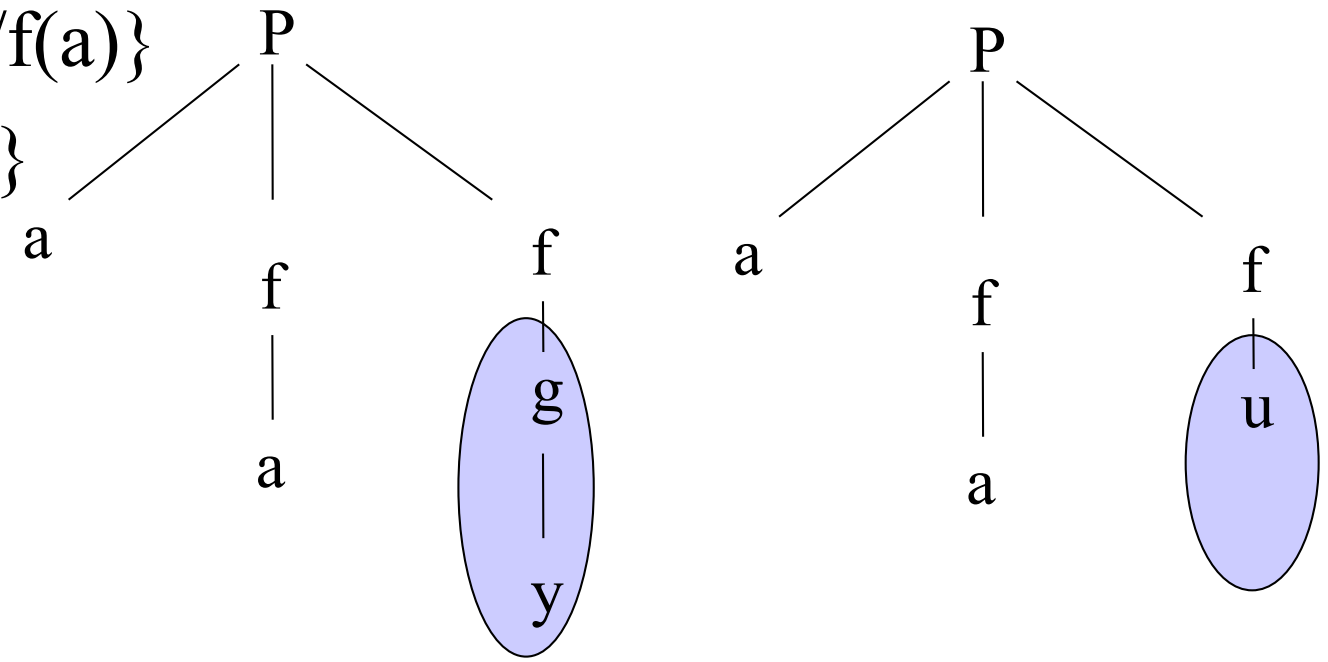
$W = \{ P(a, x, f(g(y))), P(z, f(z), f(u))\}$

$k = 2$

$W_k = \{ P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$

$\sigma_k = \{z / a, x/f(a)\}$

$D_k = \{g(y), u \}$



Unification algorithm: *example*

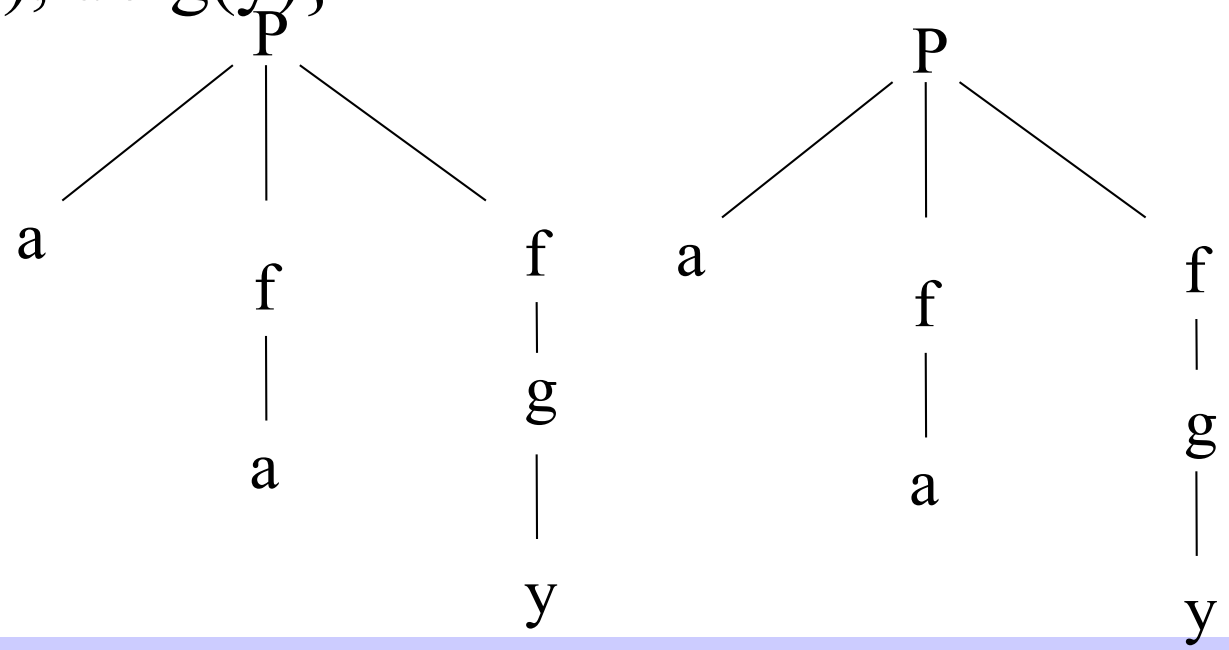
$W = \{ P(a, x, f(g(y))), P(z, f(z), f(u))\}$

$k = 3$

$W_k = \{ P(a, f(a), f(g(y))) \}$

$\sigma_k = \{z / a, x/f(a), u / g(y)\}$

$D_k = \{\}$



Unification Theorem

Theorem: if W is a non empty set of unifiable terms, then the algorithm always finishes at step 2 and the last substitution σ_k is the most general unifier of W.

L I P 6 C N R S

Automatic Theorem Proving

Tableau Method

Resolution in Propositional Logic

Unification

Resolution in First Order Logic

ACASA – Faculté des sciences de Sorbonne Université

General Resolution

Definition:

- Being given two clauses A and B of which variables are disjoint (if needed, rename variable),
- Being given a literal L_1 of A and a literal L_2 of B such that either L_1 and $\neg L_2$ or $\neg L_1$ and L_2 unify with the substitution σ. i.e. $\exists\sigma$ such that $L_1\sigma = \neg L_2\sigma$ or $L_2\sigma = \neg L_1\sigma$

C is said to be the resolvant of A and B by L_1 and L_2, (that is noted *Res(A, B; L_1, L_2)*)iff:

$$\mathbf{C = Res(A, B; L_1, L_2) = (A\sigma - \{L_1\sigma\}) \cup (B\sigma - \{L_2\sigma\})}$$

Examples:

Res($\underline{R(x, y)}$, $\underline{\neg R(u, v)}$; R(x, y), ¬R(u, v)) =∅

Res($\underline{P(x)} \vee P(y)$, $\underline{\neg P(z)} \vee \neg P(r)$; P(x), ¬P(z)) = $P(y) \vee \neg P(r)$

Res($\underline{man(Socrates)}$,$\underline{\neg man(z)} \vee mortal(z)$;man(Socrates), ¬man(z)) = mortal(Socrates)

Resolution : example

C_1: vénéneux(X) $\vee$ ¬champignon(X) $\vee$ ¬non-comestible(X)

C_2: toxique(U) $\vee$ ¬vénéneux(U)

C_3: ¬toxique(a)

Rés(C_1, C_2 ; vénéneux(X) , ¬vénéneux(U)) =

Rés(**vénéneux(X)** $\vee$ ¬champignon(X) $\vee$ ¬non-comestible(X),
toxique(U) $\vee$ **¬vénéneux(U)**
; vénéneux(X) , ¬vénéneux(U)) =

¬champignon(X) $\vee$ ¬non-comestible(X) $\vee$ toxique(X)

C_1 means **champignon(X) $\wedge$ non-comestible(X) $\Rightarrow$ vénéneux(X)**

C_2 means **vénéneux(U) $\Rightarrow$ toxique(U)**

La resolvant of C_1 and C_2 means
champignon(X) $\wedge$ non-comestible(X) $\Rightarrow$ toxique(X)

Resolution : example (following)

C_1: vénéneux(X) $\vee$ ¬champignon(X) $\vee$ ¬non-comestible(X)

C_2: toxique(U) $\vee$ ¬vénéneux(U)

C_3: ¬toxique(a)

Res(C_2, C_3 ; toxique(U) , ¬toxique(a)) =

Res(**toxique(U)** $\vee$ ¬vénéneux(U),

¬toxique(a) ,

; toxique(U) , ¬toxique(a)) =

¬vénéneux(a),

C_2 means **vénéneux(U) $\Rightarrow$ toxique(U)**, which is equivalent to **¬toxique(U) $\Rightarrow$ ¬vénéneux(U)**,

C_3 means **¬toxique(a)** (in other words that a is not toxic)

The resolvant of C_2 and C_3 means **¬vénéneux(a)** in other words that a is not venenous.

Resolution Theorem

Resolution theorem: being S a set of clauses,

S is unsatisfiable if and only if there exists a R-refutation de S,

in other words, if it is possible to derive the empty clause (□, i.e. false) by iteratively applying the resolution rule, which means that $S \vdash_{Res} \square$.

Automating the Proof

- Why?
  - Proof of programs
  - Artificial intelligence (*mathematical reasoning*)
  - Programming (*logic-based programming language*)
  - Knowledge representation (*artificial intelligence and data bases*)
- How?

How to automatically prove a theorem?

- **Proof by refutation**:

  Being S a set of clauses and C a clause. To prove that $S \vdash C$ it is enough to prove that S and $\neg C$ are unsatisfiable, which is equivalent to refutate S and $\neg C$ i.e. to proof that S, $\neg C \vdash_{Res} \square$

- **Resolution**:

$$C_k = Res(C_i, C_j; L_{i1}, L_{j2}) = (C_i\sigma - \{L_{i1}\sigma\}) \cup (C_j\sigma - \{L_{j2}\sigma\})$$

- **Choices** C_i, C_j, L_{i1}, L_{j2}
  - Complexity at step k: $\mathbf{(|S| + k)^2 * |C_i| * |C_j|}$

Horn clause

Horn clause: *not more* than a positive literal

Positive Horn clause: *exactly* one positive literal

Also called "definite clause"

Negative Horn clause: *no* positive literal

Also called "Query"

Notation: « P :- N_1, N_2, …, N_n. » is a Horn clause.

« P » is the clause **head**

« N_1, N_2, …, N_n. » is the **body** or the **goal** part of the clause

SLD-Resolution

Definition: (SLD: Selection rule, Linear resolution, Definite clauses)

Selection: choice of literals in the clauses

Linear resolution: one of the two clauses comes from the previous application of the resolution

Definite clauses: positive Horn clauses

How to automatically prove a theorem?

- **Proof by refutation**:

  Being S a set of clauses and C a clause. To prove that $S \vdash C$ it is enough to prove that S and $\neg C$ are unsatisfiable, which is equivalent to refutate S and $\neg C$ i.e. to proof that $S, \neg C \vdash_{Res} \square$

- **Resolution**:

$$C_k = Res(C_i, C_j; L_{i1}, L_{j2}) = (C_i\sigma - \{L_{i1}\sigma\}) \cup (C_j\sigma - \{L_{j2}\sigma\})$$

- **Choices** C_i, C_j, L_{i1}, L_{j2}
  - **<u>Complexity at step k</u>**: $(|S| + k)^2 * |C_i| * |C_j|$

Completeness of SLD resolution

SLD resolution is complete when the set of clauses S is composed of:

1. A negative Horn clause
2. A set of definite Horn clauses

(which are necessarily satisfiable…)

Complexity at step k: $|S|$ *instead of* $(|S| + k)^2 * |C_i| * |C_j|$

Completeness of the SLD refutation