Computational Logic

Unification and Resolution

Damiano Zanardini

UPM European Master in Computational Logic (EMCL)
School of Computer Science
Technical University of Madrid
damiano@fi.upm.es

Academic Year 2008/2009

Introduction

[R65], abstract

Theorem-proving on the computer, using procedures based on the fundamental theorem of Herbrand concerning the first-order predicate calculus, is examined with a view towards improving the efficiency and widening the range of practical applicability of these procedures. A close analysis of the process of substitution (of terms for variables), and the process of truth-functional analysis of the result of such substitutions, reveals that both processes can be combined into a single new process (called resolution), iterating which is vastly more efficient than the older cyclic procedures consisting of substitution stages alternating with truth-functional analysis stages.

The theory of the resolution process is presented in the form of a system of first-order logic with just one inference principle (the resolution principle). The completeness of the system is proved; the simplest proof-procedure based on the system is then the direct implementation of the proof of completeness. However, this procedure is quite inefficient, and the paper concludes with a discussion of several principles (called search principles) which are applicable to the design of efficient proof-procedures employing resolution as the basic logical process.

Introduction

From [R65]

- traditionally, a single step in a deduction has been required, for pragmatic and psychological reasons, to be simple enough, broadly speaking, to be apprehended as correct by a human being in a single intellectual act
- from the theoretical point of view, however, an inference principle need only to be *sound* and *effective*
- when the agent carrying out the application of an inference principle is a modern computing machine, [...] more powerful principles [...] become a possibility
- in the system described in this paper, one such inference principle is used. It is called the *resolution principle*, and it is machine-oriented, rather than human-oriented

Introduction

From [R65]

- the main advantage of the resolution principle lies in its ability to allow us to avoid one of the major combinatorial obstacles to efficiency which have plagued earlier theorem-proving procedures
  - (cited in the paper) Gilmore
  - (cited in the paper) Davis-Putnam
  - ground resolution (as presented before)

Substitutions

Formal definition

A *substitution* is a partial function (with finite domain) mapping variables to terms: $\alpha = \{\ x_1/t_1,\ x_2/t_2,\ ..,\ x_n/t_n\ \}$

- $x_1, .., x_n$ are distinct variables
- for every i, x_i does not occur in t_i

Terminology

- *bounding*: a pair x_i/t_i
- $Domain(\alpha) = \{x \mid x/t \in \alpha\}$
- $CoDomain(\alpha) = \{y \mid \exists t(\exists x(x/t \in \alpha) \wedge y \text{ occurs in } t)\}$
- $\lambda = \{\}$ (*empty substitution*)
- if α is *bijective* from variables to variables, then it is called a *renaming*

Substitutions

Examples: variables x, y, z, w

$$\alpha_1 = \{\ x/f(a),\ y/x,\ z/h(b,y),\ w/a\ \}$$

$$Domain(\alpha_1) = \{x,y,z,w\}$$
$$CoDomain(\alpha_1) = \{x,y\}$$

$$\alpha_2 = \{\ x/a,\ y/a,\ z/h(b,c),\ w/f(d)\ \}$$

$$Domain(\alpha_2) = \{x,y,z,w\}$$
$$CoDomain(\alpha_2) = \{\}$$

$$\alpha_3 = \{\ x/y,\ z/w\ \}$$

$$Domain(\alpha_3) = \{x,z\}$$
$$CoDomain(\alpha_3) = \{y,w\}$$

$$\lambda = \{\} = \{\ x/x,\ y/y,\ z/z\ \}$$

Substitutions

Application of α to F

The *application* $F\alpha$ of a substitution α to F is the formula which is obtained by replacing at the same time for all i every occurrence of x_i in F by t_i, for each $x_i/t_i \in \alpha$

$$\alpha = \{\ x/f(a),\ \ y/x,\ \ z/h(b,y),\ \ w/a\ \}$$

- $(p(x,y,z))\alpha = p(f(a), f(a), h(b, f(a))) \rightsquigarrow$ incorrect
- $(p(x,y,z))\alpha = p(f(a), x, h(b,y)) \rightsquigarrow$ correct

Terminology (2)

- F' is an *instance* of F if there exists α non-empty such that $F' = F\alpha$
- α is *idempotent* iff $((F\alpha)\alpha = F\alpha)$
  - this happens when $\mathit{Domain}(\alpha) \cap \mathit{CoDomain}(\alpha) = \emptyset$
  - $\{x/a, y/f(b), z/v\}$ is idempotent, $\{x/a, y/f(b), z/x\}$ is not

Substitutions

Composition of substitutions

Given $\alpha = \{x_1/t_1, .., x_n/t_n\}$ and $\beta = \{y_1/s_1, .., y_m/s_m\}$, the *composition* $\alpha\beta$ of these substitutions is defined as:

$$\{ \ x_1/(t_1\beta), \ .., \ x_n/(t_n\beta), \ y_1/s_1, \ .., \ y_m/s_m \ \}$$

removing the elements such that (1) $x_i \equiv t_i\beta$; or (2) $y_j \in \{x_1, .., x_n\}$

Example

$\alpha = \{ \ x/3, \ y/f(x,1) \ \}$ and $\beta = \{ \ x/4 \ \}$ give $\alpha\beta = \{ \ x/3, \ y/f(4,1) \ \}$ and $\beta\alpha = \{ \ x/4, \ y/f(x,1) \ \}$

Properties

$(F\alpha)\beta = F(\alpha\beta)$ $(f.vs.)$

$\alpha\lambda = \lambda\alpha = \alpha$

$(\alpha\beta)\gamma = \alpha(\beta\gamma)$

$\alpha\beta \neq \beta\alpha$

Unifiers

Definition

A substitution α is a *unifier* of two formulæ F and G if $F\alpha = G\alpha$

- in this case, F and G are said to be *unifiable*
- a unifier α of F and G is called *most general unifier* (*MGU*) iff for any other unifier β of F and G there exists γ such that $\beta = \alpha\gamma$
- two unifiable formulæ have only one (apart from renaming) *MGU*

Example: $F = p(x, f(x, g(y)), z)$ and $G = p(v, f(v, u), a)$

- $\alpha_1 = \{\ x/v,\ u/g(y),\ z/a\ \}$ $\quad \alpha_2 = \{\ x/a,\ v/a,\ y/b,\ u/g(b),\ z/a\ \}$
- $F\alpha_1 = G\alpha_1 = p(v, f(v, g(y)), a)$
- $F\alpha_2 = G\alpha_2 = p(a, f(a, g(b)), a)$
- α_1 and α_2 are both unifiers, but α_1 is the *MGU*: $\alpha_2 = \alpha_1\gamma$ for $\gamma = \{v/a, y/b\}$

Unification Algorithm

Several versions

- Robinson. [R65]. 1965
- Chang, Lee. Symbolic Logic and Mechanical Theorem Proving. 1973
  - a generalization of the presented version
- Martelli, Montanari. An Efficient Unification Algorithm. 1982
- Escalade-Imaz, Ghallab. A Practically Efficient and Almost Linear Unification Algorithm. 1988
- Henckel. An Efficient Linear Unification Algorithm. 1997
- Suciu. Yet Another Efficient Unification Algorithm. 2006
- and many others (the list is incomplete and inconsistent!)

This short list is enough to realize that *efficiency* is the main issue here

Unification Algorithm

Computes the *MGU* of two atoms F and G with the same predicate

$\alpha = \lambda$
while $(F\alpha \neq G\alpha)$
  find the leftmost symbol in $F\alpha$ such that
    the corresponding symbol in $G\alpha$ is different
  let t_F and t_G be the terms in $F\alpha$ and $G\alpha$ which begin with such symbols:
    if (neither t_F nor t_G are variables) or
      (one is a variable which occurs in the other one)
    then FAIL: F and G are not unifiable
    else if (t_F is a variable) **then** $\alpha = \alpha(t_F/t_G)$
    else if (t_G is a variable) **then** $\alpha = \alpha(t_G/t_F)$
α is the *MGU* of F and G

Unification Algorithm

Example: $F = p(x, x)$ and $G = p(f(a), f(b))$

α	$F\alpha$	$G\alpha$	t_F	t_G
λ	$p(x, x)$	$p(f(a), f(b))$	x	$f(a)$
$\{x/f(a)\}$	$p(f(a), f(a))$	$p(f(a), f(b))$	a	b

FAIL: F and G are not unifiable

Example: $F = p(x, f(y))$ and $G = p(z, x)$

α	$F\alpha$	$G\alpha$	t_F	t_G
λ	$p(x, f(y))$	$p(z, x)$	x	z
$\{x/z\}$	$p(z, f(y))$	$p(z, z)$	$f(y)$	z
$\{x/f(y), z/f(y)\}$	$p(f(y), f(y))$	$p(f(y), f(y))$		

F and G have a MGU: $\{\ x/f(y),\ z/f(y)\ \}$

Resolution with Unification

Rule of resolution with unification

Let $L_1 \vee F_1$ and $\neg L_2 \vee F_2$ two clauses where the literals L_1 and L_2 have the same predicate symbol. A new clause $(F_1\beta \vee F_2)\alpha$ can be deduced, such that

- β is a renaming such that $(L_1 \vee F_1)\beta$ and $\neg L_2 \vee F_2$ do not have common variables
- α is a unifier of L_1 and L_2

The new clause is called the *resolvent* of $L_1 \vee F_1$ and $\neg L_2 \vee F_2$

Rule of factorization

Given $L_1 \vee .. \vee L_n \vee F$, where L_i have the same predicate symbol, a new clause $L \vee F\alpha$ can be derived, where

- α is a unifier (maybe the *MGU*) of $L_1, .., L_n$
- $L = L_1\alpha = .. = L_n\alpha$

L is called a *factor* of $L_1 \vee .. \vee L_n \vee F$

Resolution with Unification

"Resolution with Unification" (RU) step

Apply the rule of factorization, followed by resolution with unification

- in the system described in this paper, one such inference principle is used. It is called the *resolution principle*, and it is machine-oriented, rather than human-oriented [R65]

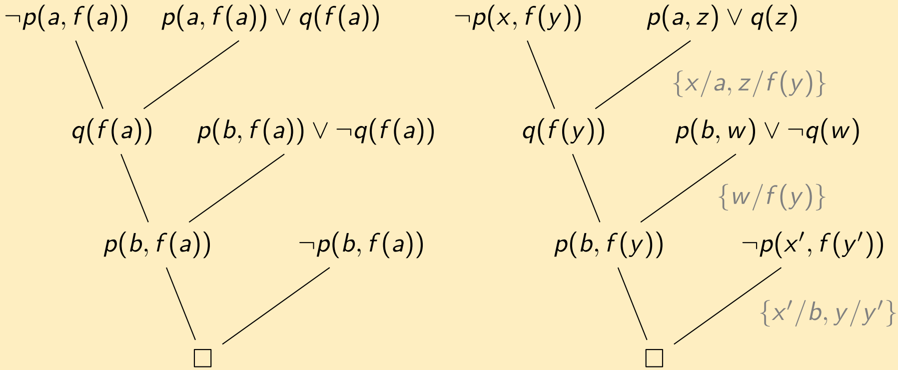
The method

It is possible to build resolution trees where the resolvent of each two clauses can be obtained by an RU step

- for every step of *ground resolution* there is a step of *resolution with unification*

Resolution with Unification

$C_1 = \neg p(x, f(y)),\ C_2 = p(a, z) \vee q(z),\ C_3 = p(b, u) \vee \neg q(u)$



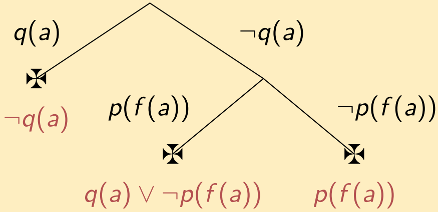
ground instance resolution

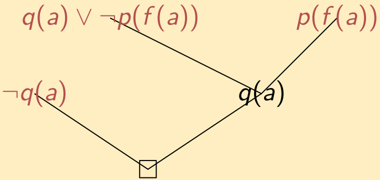
resolution with unification

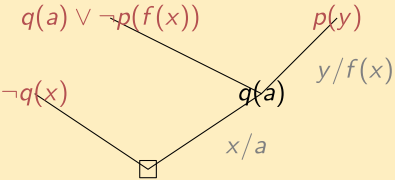
Resolution with Unification

Semantic trees vs. resolution trees

$$
\begin{aligned}
C_1 &= p(y) \\
C_2 &= q(a) \vee \neg p(f(x)) \\
C_3 &= \neg q(x)
\end{aligned}
$$







Resolution with Unification

Lemma (Lifting Lemma)

Let C_1 (resp., C_2) be a clause and B_1 (resp., B_2) one of its ground instances. If B is a resolvent of B_1 and B_2, then

- *there exists a clause C which has B as one of its ground instances*
- *C results from a resolution step on C_1 and C_2 w.r.t. a literal L which is a common factor of C_1 and C_2:*

$$C_1 = L_1 \vee .. \vee L_n \vee D_1 \qquad\qquad C_2 = \neg L_{n+1} \vee .. \vee \neg L_{n+m} \vee D_2$$
$$C = (D_1\rho_1 \vee D_1\rho_2)\theta$$

- *ρ_1 and ρ_2 are renamings such that $C_1\rho_1$ and $C_2\rho_2$ have no common variables (actually, one renaming is enough)*
- *θ is a MGU: $L_1\rho_1\theta = .. = L_n\rho_1\theta = L_{n+1}\rho_2\theta = .. = L_{n+m}\rho_2\theta = L$*

Resolution with Unification

MGU resolution rule

Let

$$C_1 = L_1 \vee .. \vee L_n \vee D_1 \qquad\qquad C_2 = \neg L_{n+1} \vee .. \vee \neg L_{n+m} \vee D_2$$

where all L literals have the same predicate symbol. A new clause

$$(D_1\rho_1 \vee D_2\rho_2)\theta$$

can be deduced, where

- ρ_1 and ρ are renamings:

$$\begin{array}{c} Domain\,(\rho_1) = Vars(C_1) \\ Domain\,(\rho_2) = Vars(C_2) \\ CoDomain\,(\rho_1) \cap CoDomain\,(\rho_2) = \emptyset \end{array}$$

- θ is the *MGU* of $L_1\rho_1, \ldots, L_n\rho_1, L_{n+1}\rho_2, \ldots, L_{n+m}\rho_2$

Resolution with Unification

Lemma (*MGU* resolution rule, correctness)

$[\ \forall x_1..x_p C_1,\ \ \forall y_1..y_q C_2\ ] \vdash \forall z_1..z_r((D_1\rho_1 \vee D_2\rho_2)\theta)$ *is correct, where*

- $\{x_1,..,x_p\} = Vars(C_1)$, $\{y_1,..,y_q\} = Vars(C_2)$, $\{z_1,..,z_r\} = Vars((D_1\rho_1 \vee D_2\rho_2)\theta)$
- ρ_1 *is a renaming of* $x_1..x_p$ *and* ρ_2 *is a (disjoint from* ρ_1*) renaming of* $y_1..y_q$
- θ *is the MGU of* $L_1\rho_1, \ldots, L_n\rho_1, L_{n+1}\rho_2, \ldots, L_{n+m}\rho_2$

Proof.

❶ $\forall x_1..x_p(L_1 \vee .. \vee L_n \vee D_1)$ — hypothesis ($C_1 = \overline{L} \vee D_1$)

❷ $\forall z_1..z_r(\neg L_{n+1} \vee .. \vee \neg L_{n+m} \vee D_2)$ — hypothesis ($C_2 = \overline{\neg L} \vee D_2$)

❸ $F \vee E_1$ — apply ρ_1 and θ to C_1, idempotence $F \vee .. \vee F = F$

❹ $\neg F \vee E_2$ — apply ρ_2 and θ to C_2, idempotence $\neg F \vee .. \vee \neg F = \neg F$

❺ $E_1 \vee E_2$ — cut on ❸ and ❹

❻ $\forall z_1..z_r((D_1\rho_1 \vee D_2\rho_2)\theta)$ — generalization of ❺

Resolution with Unification

Lemma

Let $\mathcal{C}$ be an unsatisfiable set of clauses with a closed semantic tree of depth $n \geq 1$. Then there is a set R of resolvents of $\mathcal{C}$ such that $\mathcal{C}' = \mathcal{C} \cup R$ has a closed semantic tree of depth $n - 1$

Proof.

1. let B_1, B_2 be two ground instances of $C_1, C_2 \in \mathcal{C}$ which are false in two failure nodes (brothers) at level n (the deepest in the tree)
2. the resolvent of B of B_1 and B_2 is false in the parent node (depth $n - 1$)
3. by the Lifting Lemma, there exists an *MGU* resolvent C of C_1 and C_2 such that B is a ground instance of C
4. let R be the set of such Cs, obtained by considering all pairs of failure nodes at the maximum depth n
5. a closed semantic tree of $\mathcal{C} \cup R$ can be constructed which has maximum depth $n - 1$ (essentially, by pruning the initial tree)

Resolution with Unification

Lemma (*MGU* resolution)

If a set $\mathcal{C}$ of clauses is unsatisfiable, then $\Box$ is deduced from it by MGU resolution

Proof.

1. $UNSAT(\mathcal{C})$
2. there exists an n-deep closed semantic tree (by Herbrand's Theorem)
3. if $n = 0$ (only the root), then $\Box \in \mathcal{C}$: trivial
4. if $n > 1$, then
   - there exists a set R of resolvents of $\mathcal{C}$ clauses, such that $\mathcal{C}' = \mathcal{C} \cup R$ has a $(n-1)$-deep closed semantic tree (by the Lemma above)
   - the rest follows by induction

Resolution with Unification

Theorem (*MGU* resolution)

A set $\mathcal{C}$ of clauses is unsatisfiable iff $\Box$ can be deduced from it by MGU resolution ($\mathcal{C} \vdash_{MGU} \Box$)

Proof ($\rightarrow$).

Follows by the *MGU* resolution Lemma

Proof ($\leftarrow$).

1. $\mathcal{C} \vdash \Box$ by *MGU* resolution
2. $\mathcal{C} \models \Box$ for the correctness of *MGU* resolution
3. $\Box$ is false in every interpretation
4. $\mathcal{C}$ must be false in every interpretation
5. $UNSAT(\mathcal{C})$

Method of Saturation

Let $\mathcal{C}$ be a set of clauses

$S_0 = \mathcal{C}$
$n = 0$
repeat
  if $(\Box \in S_n)$ **then STOP**: $UNSAT(\mathcal{C})$
  else
    $S_{n+1} = \{\text{resolvents of } C_1 \text{ and } C_2 \mid C_1 \in S_1 \cup .. \cup S_n,\ C_2 \in S_n\}$
    if $(S_{n+1} = \emptyset)$ or $(S_{n+1} \sqsubseteq S_1 \cup .. \cup S_n)$ **then STOP**: $SAT(\mathcal{C})$
    $n = n + 1$

Completeness: $UNSAT(\mathcal{C})$ iff $\Box$ is derived

- the construction of S_{n+1} requires considering all possible factors of C_1 and C_2
- this method generates *all* and *only* the resolvents of $\mathcal{C}$ clauses
- a number of redundant clauses are generated

Method of Saturation

Example: $\mathcal{C} = \{p \vee q,\ \neg p \vee q,\ p \vee \neg q,\ \neg p \vee \neg q\}$

$S_0 =$ (1) $p \vee q$
(2) $\neg p \vee q$
(3) $p \vee \neg q$
(4) $\neg p \vee \neg q$

$S_1 =$ (5)	q	(1,2)
(6)	p	(1,3)
(7)	$q \vee \neg q$	(1,4)
(8)	$p \vee \neg p$	(1,4)
(9)	$q \vee \neg q$	(2,3)
(10)	$p \vee \neg p$	(2,3)
(11)	$\neg p$	(2,4)
(12)	$\neg q$	(3,4)

even after one step there are redundant and tautological clauses

Method of Saturation

Conclusion

- *MGU* resolution allows to decide satisfiability without the need to use ground instances
- however, saturation is not efficient since it generates many useless clauses
  - the raw implementation of the Resolution Principle would produce a very inefficient refutation procedure [R65]
  - by Church's Theorem we know that for some inputs S this procedure, and in general all correct refutation procedures, will not terminate [R65]

Example [R65]

$$C_1 = q(a) \qquad\qquad C_2 = \neg q(x) \vee q(f(x))$$

at any step $q(f^n(a))$ is generated, for n increasing by 1 each time