A MECHANIZATION OF TYPE THEORY

Gérard P. HUET
IRIA - LABORIA
Rocquencourt FRANCE

ABSTRACT

A refutational system of logic for a language of order ω is presented. This langage is a slight modification of Church's λ-calculus with types. The system is complete, in the sense that a refutation of a set of sentences exists if and only if this set does not possess a general Henkin model. The main rule of inference is a generalization of Robinson's resolution to type theory, which allows us to get rid of the substitution rule.

INTRODUCTION

During the last decade, a lot of effort in automatic theorem proving has been devoted to mechanizing first-order logic. The main achievement has been the definition by J.A.Robinson[14] of a now well-known refutational first-order system. Its unique rule of inference, called resolution, is an elegant way of combining the substitution and cut rules. The other familiar rules of inference are implicitly taken care of by Skolemization and the set representation (clauses) of the conjunctive normal form of the set of sentences one wants to refute. More precisely, resolution permits us to select just the substitutions which are necessary for the cut of two or more literals. Moreover, if this cut is possible at all, only one such substitution is sufficient, called the most general unifier of the two literals. This existence of a most general unifier between two terms in first-order logic is fundamental to the resolution method.

Many a theorem proving program was written based on the resolution rule, embodying various heuristics to speed up the search for a refutation. However, results obtained so far are still very limited without human help. To remedy this situation, some workers in the field, notably Robinson[16] have argued for the exploration of more powerful systems of logic such as Church's λ-calculus.

HIGHER ORDER UNIFICATION

The situation in higher-order logic is quite different from first-order. Most general unifiers do not exist any more (Gould[8]).

Consider for instance the two terms :

$$t_1 = f(x)$$

and

$$t_2 = A$$

where f is a (second-order) variable, x a variable and A a constant. The following substitutions are two independent unifiers of t_1 and t_2 :

$$\sigma_1 = \{f \leftarrow \lambda u \cdot A\}$$

$$\sigma_2 = \{f \leftarrow \lambda u \cdot u,\ x \leftarrow A\}$$

We may even need to consider an infinity of unifiers. Let us say that substitution σ is less general than substitution ρ with respect to a finite set of variables V, if there exists η such that :

$$\sigma \circ x = \eta \circ \rho \circ x \quad \text{for every } x \text{ in } V .$$

We shall denote this relation (which is reflexive and transitive) by : $\rho \leqslant_V \rho$.

Now we call a complete generator of unifiers for terms t_1 and t_2 (CGU in short) any set Σ of substitutions such that :

$$\begin{cases} 1^\circ \quad \forall\, \sigma \in \Sigma : \sigma \text{ unifies } t_1 \text{ and } t_2 . \\ 2^\circ \quad \forall\, \sigma \text{ unifier of } t_1 \text{ and } t_2 : \exists \rho \in \Sigma \text{ such that } \sigma \leqslant_V \rho \text{, where } V \text{ contains all variables free in } t_1 \text{ and } t_2 . \end{cases}$$

Gould[8] has remarked that certain pairs of terms t_1, t_2 do not possess a finite CGU. Consider for instance :

$$\begin{cases} t_1 = f(x, A) \\ t_2 = f(x, B) \end{cases}$$

where x is a (variable) function and f is a (variable) functional. Now let $\Sigma = \{\sigma_i \mid i \geqslant 0\}$, with :

$$\sigma_0 = \{f \leftarrow \lambda uv \cdot h(u)\}$$

$$\sigma_n = \{f \leftarrow \lambda uv \cdot g_n(u, u(h_1^n(u,v)), \ldots, u(h_n^n(u,v))),\ x \leftarrow \lambda w \cdot z\} \qquad (n > 0)$$

and let

$$V = \{x, f\} .$$

The proof can be sketched as follows :

1° Σ is a CGU for t_1 and t_2

2° $\sigma_i \leqslant_V \sigma_{i+1}$ $\quad \forall\, i > 0$

3° not $\sigma_{i+1} \leqslant_V \sigma_i$ $\quad \forall\, i > 0$

4° not ($\sigma_0 \underset{V}{\leq} \sigma_i$ or $\sigma_i \underset{V}{\leq} \sigma_0$) $\forall\, i > 0$.

5° suppose Σ' is a finite CGU for t_1 and t_2

By 1°, this implies that there exists $\Sigma'' \subset \Sigma$ finite CGU for t_1 and t_2. This obviously contradicts 3° and 4°. (Consider σ_{k+1}, with $k = \max\{i \mid \sigma_i \in \Sigma''\}$).
Thus, although any finite subset of Σ can be generated by only two elements (using 2°), we cannot find a finite set of generators for the unifiers of t_1 and t_2.

It seems therefore that an extension of resolution to type theory would require generating a possibly infinite number of resolvents at every application of the resolution rule. One solution to this problem would be to order these resolvents (for instance according to some complexity of the unifier selected) and to "dovetail" the generation of resolvents with the refutation search algorithm. This method has been proposed and proved complete by Pietrzykowski and Jensen[13].

However the inefficiency of such dovetailing processes is well known and their practical implementation seems difficult. We shall present here a method which, although it ultimately leads to a double enumeration too, delays the generation of unifiers as much as possible and thus minimizes the number of irrelevant choices.

CONSTRAINTS

The idea is to represent the set of unifiers of two literals L_1 and L_2 as a condition tagging the resolvent obtained by cutting L_1 and L_2 in their respective clauses. Such a condition is expressed as a set of "constraints" of the form $\{t_1, t_2\}$ where t_1 and t_2 are subterms of L_1 and L_2. More generally, a constraint will be a set of terms which we want to unify. A clause tagged by a set of constraints will be represented as

$$L_1, L_2, \ldots, L_n / C_1, C_2, \ldots, C_p$$

where the L_i's are the literals of the clause and the C_i's are constraints. Such a "constrained clause" (CC in short) represents the set of all clauses $\sigma \circ L_1, \sigma \circ L_2, \ldots, \sigma \circ L_n$ such that σ unifies simultaneously every constraint C_i ; i.e., if $C_i = \{e_1, e_2, \ldots, e_{q_i}\}$ then
$\sigma \circ e_1 = \sigma \circ e_2 = \ldots = \sigma \circ e_{q_i}$ $(1 \leq i \leq p)$.

In many cases we will be able to decide if a constraint is not unifiable at all, in which case the corresponding CC is not generated. We may also know a most general unifier σ for the constraint, in which case σ is applied to the corresponding resolvent, tagged with the union of the constraints in the CCs resolved upon. Otherwise we generate the CC obtained by tagging the resolvent with the union of the parent's constraints plus the new one. In the last two cases we check that the set of constraints is not obviously inconsistent. Let us now give an example.

We suppose that we have two basic types : o for truth values and ι for individuals. We shall use the following atoms, indicated with their type :

w, A	ι
v, y, B, C	$(\iota \to \iota)$
x, z, u	$((\iota \to \iota) \to \iota)$
f	$(((\iota \to \iota) \to \iota) \to \iota)$
P, R	$(\iota \to o)$
Q	$(\iota, (\iota \to \iota) \to o)$

By convention, all lower case letters denote variables, all capitals denote constants. $(\alpha_1, \alpha_2, \ldots, \alpha_n \to \beta)$ denotes the type of functions of n^2 arguments of types $\alpha_1, \ldots, \alpha_n$ and with values of type β. We consider the set S of clauses :

1° P(f(x)), Q(z(C), C), R(f(z))
2° ¬P(A)
3° ¬Q(y(A), y)
4° ¬R(B(w))

We shall prove by constrained resolution that this set is unsatisfiable (the precise statement of the rule of constrained resolution will be given below).
First we resolve the P's in 1° and 2°. We are trying to unify P(f(x)) and P(A), and this implies the unification of f(x) and A. These terms do not possess a most general unifier, so we generate a constraint {f(x), A} which is tagged to the resolvent :

5° Q(z(C), C), R(f(z))/{f(x), A}.

Now we resolve the Q's in 5° and 3°, which implies the simultaneous unification of :

firstly y(A) and z(C)
and secondly y and C .

Fortunately in the second pair {y ← C} is a most general unifier. This unifier transforms the first condition into the unification of C(A) and z(C). Here too there is no MGU, so we generate the resolvent :

6° R(f(z))/{f(x), A}, {C(A), z(C)}.

Finally we get the empty clause, resolving 6° and 4°, and generating one more constraint {B(w), f(z)} :

7° ∅/{f(x), A}, {C(A), z(C)}, {B(w), f(z)}.

Now, in order to validate our refutation, we must find a unifier satisfying simultaneously every constraint in 7°. However any unifier will do here, we do not need to generate a complete set of them. Actually, we just need to check for the existence of a unifier. Unfortunately there is no decision procedure for this task if we allow quantification over third-order functionals, as shown in Huet[11].

All we can do here is to use a semi-decision algorithm of reasonable efficiency developed for this problem in Huet[10] . We shall not present this algorithm here, but merely indicate that for our example it would return an affirmative answer, with the unifier :

$$\begin{cases} f \leftarrow \lambda u \cdot u(B) \\ z \leftarrow \lambda v \cdot v(A) \\ w \leftarrow A \\ x \leftarrow \lambda v \cdot A \end{cases}$$

The reader will check that this substitution (together with $y \leftarrow C$) applied to S gives a contradictory set of ground clauses.

The general case is a little more complicated because, when resolving two CCs, each of which has a non-empty set of constraints, we have to merge these two sets.

Note also that when we apply a substitution to a CC we must apply it to its constraints as well, which may induce some possible simplifications, merging of constraints and rejections.

A more serious complication arises with the use of predicate variables. A literal whose predicate is a variable may be transformed into several literals by substitution of a disjunction for the variable. For instance the clause $p(A), S(B)$ becomes

$$q(A), r(A), S(B)$$

after the substitution

$$p \leftarrow \lambda u \cdot (q(u) \vee r(u))$$

and passage to normal form. Similarly, the substitution of a corresponding conjunction would produce two clauses :

$$q(A), S(B)$$

and

$$r(A), S(B) .$$

Also, substituting a quantified formula for p would oblige us to Skolemize during the proof.

For these reasons, we are obliged to introduce a second rule of inference, called splitting, which in effect simulates all possible ways of effecting these substitutions for predicate variables, using the mechanism of the constraints.

We shall now give a more precise formulation of our rules of inference.

REFUTATIONS

We shall not state formally the definitions of terms, literals, etc... The precise definitions are stated in Huet[10,11] . Let us just recall that our language is basically Church's λ-calculus with types (Church[3]). Every term possesses a normal form :

$$\lambda u_1 \ldots u_n \cdot \sigma(e_1, \ldots, e_p)$$

where the u_i's are distinct variables, σ is an atom (called the head of the term) and the e_j's are terms (see Andrews'). Functional extensionality is an option in the system. We shall distinguish three special constants $\neg$ $\vee$ and π of the appropriate types. They will denote respectively negation, disjunction and universal quantification. For the last one, we represent $\forall x \cdot \sigma$ by $\pi(\lambda x \cdot \sigma)$. Actually for each type α there is a π_α of type $((\alpha \to o) \to o)$, denoting quantification over variables of type α .

We suppose moreover that the set of sentences one wants to refute has been reduced to clause form. This means in particular that we have Skolemized as much as we could. Of course, it is forbidden to reduce propositions which are hidden inside the arguments of some literal. Our Skolemization consists in replacing (at the top level) :

$\neg\pi_\alpha(\sigma)$ by $\neg\sigma(x)$ where x is a new variable of type α

$\neg\neg\pi_\alpha(\sigma)$ by $\neg\sigma(E_\alpha(\sigma))$ where E_α is a special constant (called a parameter) of type $((\alpha \to o) \to \alpha)$.

After reduction, every clause will be represented as a finite set of literals, each literal being of the form A or $\neg A$, where A is a term $P(e_1, e_2, \ldots, e_n)$ and P is an atom different from $\neg$ $\vee$ and π.

Let us now give more precisely the definition of CC.

A <u>constraint</u> is any finite set $\{e_1, e_2, \ldots, e_n\}$ of terms of the same type. A substitution σ is said to unify this constraint if and only if σ is a unifier for $e_1, e_2, \ldots, e_n$. It is said to unify a set of constraints $\{C_1, C_2, \ldots, C_p\}$ if and only if it unifies every C_i $(1 \leqslant i \leqslant p)$. A <u>constrained clause</u> E/C consists of a clause E and of a finite set of constraints C . If $C = \emptyset$ we have an initial CC, if $E = \emptyset$, we have a terminal CC, $\Box = \emptyset/\emptyset$ is the empty CC.

In order to refute a set of sentences S , we first reduce it to clause form as explained above, then we tag to every clause the empty set of constraints $\emptyset$.

A refutation is any derivation from this set of initial CCs, using the three rules of inference defined below and ending with the empty CC. If such a refutation exists, we say that S is refutable.

The first two rules of inference, constrained resolution and splitting, apply to non-terminal CCs whereas the third one, unification, applies to terminal CCs. Constrained resolution is binary, splitting and unification are unary rules.

1. <u>Constrained resolution</u>.

Let $E_1 = A_1/B_1$ and $E_2 = A_2/B_2$ be two non-terminal CCs.

Let $\{x_1, x_2, \ldots, x_n\}$ be all the variables that appear free in both E_1 and E_2 , and $\{w_1, w_2, \ldots, w_n\}$ be new variables of the same types which do not appear in either E_1 or E_2. We define the substitution :

$$\theta = \{x_i \leftarrow w_i \mid 1 \leqslant i \leqslant n\} .$$

Now we distinguish p positive literals in A_1 and q negative literals in A_2. That is, using $\cup$ to denote disjoint union :

$A_1 = \{M_1, M_2, \ldots, M_p\} \cup A_3 \quad p \geq 1$

$A_2 = \{\neg N_1, \neg N_2, \ldots, \neg N_q\} \cup A_4 \quad q \geq 1$

Now let

$B_3 = \{\{M_1, M_2, \ldots, M_p, \theta \circ N_1, \theta \circ N_2, \ldots, \theta \circ N_q\}\}$

We say that the CC :

$E_3 = A_3 \cup \theta \circ A_4 / B_1 \cup \theta \circ B_2 \cup B_3$

follows by constrained resolution from CCs E_1 and E_2.

2. Splitting.

Let $E_1 = A/C$ be any non-terminal CC, such that A contains a literal L whose predicate is a variable ; i.e. $A = B \cup \{L\}$. We have the following cases :

1° if L is a positive literal :

1-1 Deduce $E_2 = B \cup \{q, r\}/C \cup \{\{L, qVr\}\}$
where q and r are new variables of type o.

1-2 Deduce $E_2 = B \cup \{\neg q\}/C \cup \{\{L, \neg q\}\}$
where q is a new variable of type o.

1-3 Deduce $E_2 = B \cup \{q(z)\}/C \cup \{\{L, \pi_\alpha q\}\}$
where z is a new variable of type α, q a new variable of type $(\alpha \to o)$.

2° if L is a negative literal ; i.e. $L = \neg M$:

2-1 Deduce $E_2 = B \cup \{\neg q\}/C_1$
and $E_3 = B \cup \{\neg r\}/C_1$
where $C_1 = C \cup \{\{M, qVr\}\}$, q and r are new variables of type o.

2-2 Deduce $E_2 = B \cup \{q\}/C \cup \{\{M, \neg q\}\}$
where q is a new variable of type o.

2-3 Deduce $E_2 = B \cup \{\neg q(E_\alpha(q))\}/C \cup \{\{M, \pi_\alpha q\}\}$
where q is a new variable of type $(\alpha \to o)$.

Any of the above CCs is said to be deducible from E_1 by splitting.

Rules 1-3 and 2-3 depend on some arbitrary type α. In practice we shall implement them by restricting α to some small initial segment of types.

3. Unification.

Let $E = \phi/C$ be a terminal CC. If C is unifiable, we can deduce $\square$ from E.

This rule too is not directly mechanizable, since in the general case unification is not decidable. We can make it so by putting a bound on the number of steps of our unification algorithm. For this reason our method may ultimately lead to a double enumeration.

SOUNDNESS AND COMPLETENESS

It is well known that higher-order logic is not complete, under the usual semantic characterization (Gödel models). However, Henkin[3] has shown that a different characterization yields completeness in ω-order logic. The trick is to interpret predicate variables of type $(\alpha \to o)$ (for instance) over domains which may be proper subsets of $2^{\mathcal{D}_\alpha}$, where $\mathcal{D}_\alpha$ is the domain of interpretation of objects of type α. We only impose that these domains be closed under operations corresponding to the different constructs of the language (for instance complementation, projection, etc...). In the same way $\mathcal{D}_{(\alpha \to \beta)}$ may be a proper subset of $\mathcal{D}_\alpha \to \mathcal{D}_\beta$.

This enlarged semantic characterization (general Henkin models) permits us to restrict validity to just those formulas which are derivable in the logic.

In the following, we shall therefore use satisfiable as an abbreviation for "has a general Henkin model".

Definition : A property Γ of finite sets of propositions is called an analytic consistency property if and only if, for any finite set S of propositions, we have (γ_1) to (γ_7) below.

- (γ_1) if $\Gamma(S)$, then there is no literal σ such that $\sigma \in S$ and $\neg\sigma \in S$.
- (γ_2) if $\Gamma(S \cup \{\sigma\})$, then $\Gamma(S \cup \{\sigma'\})$, where σ' is the λ normal form of σ.
- (γ_3) if $\Gamma(S \cup \{\neg\neg\sigma\})$, then $\Gamma(S \cup \{\sigma\})$.
- (γ_4) if $\Gamma(S \cup \{\sigma V \beta\})$, then $\Gamma(S \cup \{\sigma\})$ or $\Gamma(S \cup \{\beta\})$.
- (γ_5) if $\Gamma(S \cup \{\neg(\sigma V \beta)\})$, then $\Gamma(S \cup \{\neg\sigma, \neg\beta\})$.
- (γ_6) if $\Gamma(S \cup \{\pi_\alpha \sigma\})$, then for every term β of type α, $\Gamma(S \cup \{\pi_\alpha \sigma, \sigma(\beta)\})$.
- (γ_7) if $\Gamma(S \cup \{\neg\pi_\alpha \sigma\})$, then $\Gamma(S \cup \{\neg\sigma(X)\})$ for any constant or variable X of type α which does not occur free in σ or S.

We are interested in analytic consistency properties because they provide a sufficient condition for satisfiability, as expressed by the following lemma from Andrews[1], which extends results by Smullyan.

Lemma I. If Γ is any analytic consistency property and S is a finite set of propositions such that $\Gamma(S)$, then S is satisfiable.

For any set $\mathcal{C}$ of propositions, let $\mathcal{C}'$ be the set of sentences (closed propositions) obtained from $\mathcal{C}$ by replacing free variables by new constants in a one to one fashion. Obviously if $\mathcal{C}$ is unsatisfiable then $\mathcal{C}'$ is unsatisfiable too. Now we define the following property Γ of sets of propositions :

$\Gamma(\mathcal{C})$ if and only if $\mathcal{C}'$ is not refutable.

Lemma II. Γ is an analytic consistency property. The proof of this lemma is given in Huet[10]. The idea is to prove each of (γ_1) to (γ_7) by proving the contrapositive statement about refutations. For instance, to prove (γ_1), let us assume that there exists a literal σ such that $\sigma \in S$ and $\neg\sigma \in S$. This implies that in the clause form of S' there are clauses $\{\sigma'\}$ and $\{\neg\sigma'\}$. Therefore from S' we can deduce CC $\phi/\{\{\sigma', \sigma'\}\}$, from which we deduce $\square$ by trivial unification. S' being refutable, we have not $\Gamma(S)$ by definition of Γ. Taking the contrapositive statement

we get (γ_1).

Now let S be any unsatisfiable set of sentences. By Lemma I and Lemma II : not $\Gamma(S)$ and by definition of Γ, S is refutable.

This gives us the completeness of our system:

Theorem 1. Let S be any finite set of sentences. If S is unsatisfiable then it is refutable.

The soundness is obtained by mapping refutations in our system into refutations in Andrews' $\mathcal{R}$ system (Andrews[1]), which is known to be sound if one assumes the axiom schema of choice. Let us call axiom of choice for type α the formula:

$$\exists i \cdot \forall p \cdot [(\exists x \cdot p(x)) \supset p(i(p))]$$

where the types of the variables are :

$$\tau(x) = \alpha \ , \ \tau(p) = (\alpha \to o) \text{ and } \tau(i) = ((\alpha \to o) \to \alpha)$$

Theorem 2. Let S be any finite set of sentences. If S is refutable, then there exists a finite set $\mathcal{C}$ of axioms of choice such that $S \cup \mathcal{C}$ is unsatisfiable.

Remarks.

In our formal presentation of the rules of inference, we have supposed that we were not trying to unify at all when resolving, instead we just generated constraints. The processing of the literals which are candidates for unification and the checking for consistency of the constraints obtained is not necessary in theory, and it was more convenient to use this formalism for our proofs. However, as was shown in our example above, it is highly desirable in practice to do some amount of processing when applying constrained resolution. Of course, this does not affect the soundness or completeness of the modified system, as long as we suppress only clauses with ununifiable constraints. There is a trade-off here between many useless but fast derivations and few useless but slow ones, and it is difficult to estimate how much processing of constraints should be done until the system is implemented.

In the case where the set of initial clauses is first-order, it is possible to process completely every constraint, and splitting is not applicable. Our modified system reduces exactly to resolution in this case.

It should be noted that we have not assumed extensionality so far since Henkin models may not be extensional, as shown by Andrews[2] . However we have in option a unification algorithm which assumes the weak extensionality axiom :

$$f = \lambda x \cdot f(x) \quad .$$

We shall now present a few examples of refutations. We have replaced the cumbersome existential parameters $E(\lambda x \cdot \mathcal{A})$ by the usual Skolem functions $X(y_1, \ldots, y_n)$, where the y_i's are the free variables of $\mathcal{A}$ and X is a new constant of the appropriate type.

EXAMPLES

Example 1.

Cantor's theorem : N^N is not denumerable.

Let ι be the type "integer", n,f and h be variables, with types $\tau(n) = \iota$, $\tau(f) = (\iota \to \iota)$ and $\tau(h) = (\iota, \iota \to \iota)$.

We shall refute the negation of Cantor's theorem : "it is possible to enumerate the functions of integer to integer", i.e.

$$\exists h \ \forall f \ \exists n \ h(n) = f \ .$$

After reduction to normal form we get, using N and H as Skolem functions, $\tau(N) = ((\iota \to \iota) \to \iota)$, $\tau(H) = (\iota, \iota \to \iota)$:

1 $\quad HNf = f$

We shall need the following property of equality :

2 $\quad \neg f = g, \quad fn = gn \quad [\tau(g) = (\iota \to \iota)]$

and the existence of a successor function S satisfying :

3 $\quad \neg n = Sn \quad [\tau(S) = (\iota \to \iota)]$.

The refutation goes as follows :

R(1,2)	4	$H(Nf,n) = fn$
R(4,3)	5	$\not\Box \{fn, SH(Nf,n)\}$
U(5)	6	$\Box$ by substitution : $\{f \leftarrow \lambda u \cdot SH(u,u), \quad n \leftarrow N(\lambda u \cdot SH(u,u))\}$

Note that the constraints coming from the first constrained resolution can be completely processed, so that $C_4 = \emptyset$. This short proof (3 steps) is due to a concise representation of the problem. In particular, the use of type ι for integers gives us implicitly restricted quantification over integers without having to carry additional literals.

Example 2.

The pigeonhole principle : if we distribute m objects into n holes, with $m > n$, there is at least one hole which contains more than one object.

We choose type α for the set A of objects and type β for the set B of holes. The hypothesis is :

$$|B| < |A| < \omega$$

1° $|B| < |A|$ (finite sets) : there exists a one-one mapping from B to A, not onto.

2° $|A| < \omega$: every one-one mapping from A to A is onto.

The conclusion is : there is no one-one mapping from A to B.

Using the following atoms :

$\tau(F) = (\beta \to \alpha)$

$\tau(G) = (\alpha \to \beta)$

$\tau(h) = (\alpha \to \alpha)$

$\tau(u) = \tau(v) = \beta$

$\tau(E) = \tau(x) = \tau(y) = \alpha$

$\tau(X) = \tau(Y) = ((\alpha \to \alpha) \to \alpha)$

$\tau(Z) = ((\alpha \to \alpha), \alpha \to \alpha)$

$\tau(=) = (\alpha, \alpha \to o)$ or $(\beta, \beta \to o)$ according to the context, we get, after reduction to normal form :

1 $u = v$, $Fu \neq Fv$ } F is one-one $B \to A$
2 $Fu \neq E$ } not onto

3 $Xh \neq Yh$, $hZ(h,x) = x$ } every one-one h $A \to A$
4 $hXh = hYh$, $hZ(h,x) = x$ } is onto

5 $x = y$, $Gx \neq Gy$ G is one-one $A \to B$

We get the following refutation :

R(3,2) 6 $Xh \neq Yh / \{hZ(h,E), Fu'\} = C_6$
R(4,2) 7 $hXh = hYh / C_6$
R(7,1) 8 $u = v / C_6 + \{hXh, Fu\} + \{hYh, Fv\}$
R(8,5) 9 $x = y / C_6 + \{hXh, FGx\} + \{hYh, FGy\}$
R(9,6) 10 $\phi / \{hZ(h,E), Fu'\}, \{hXh, FGXh'\}, \{hYh, FGYh'\}, \{h'Z(h',E), Fu''\}$
U(10) 11 $\square$ by substitution : $\{h \leftarrow \lambda x \cdot FGx,\ u' \leftarrow GZ(\lambda x \cdot FGx, E),$ h' same as h , u" same as u'$\}$.

When the theorem prover gets a refutation, it is trivial to give the user a more natural proof in Andrews' system, chaining back the substitutions.
We shall demonstrate this technique here ; now S(n) means substitution in clause n and C(n,m) means cut of clause n and clause m. H is an abbreviation for $\lambda x \cdot FGx$.

S(2) 6' $FGZ(H,E) \neq E$
S(3) 7' $XH \neq YH$, $FGZ(H,E) = E$
C(6',7') 8' $XH \neq YH$
S(4) 9' $FGXH = FGYH$, $FGZ(H,E) = E$
C(6',9') 10' $FGXH = FGYH$
S(1) 11' $GXH = GYH$, $FGXH \neq FGYH$
C(10',11') 12' $GXH = GYH$
S(5) 13' $XH = YH$, $GXH \neq GYH$
C(12',13') 14' $XH = YH$
C(8',14') 15' $\boxtimes$

Example 3.

We shall now give a completely detailed refutation of a less trivial proposition. We are going to prove Knaster-Tarski's theorem : "Any function monotone over a complete lattice possesses a fixpoint". Let $(A, \leqslant)$ be the complete lattice considered.

We use the types : o for truth values
ι for the elements of A.

We represent a subset of A by its characteristic function, of type $(\iota \to o)$. We use the following axioms :

$(\forall xyz)[x \leqslant x \ \&\ ((x \leqslant y \ \&\ y \leqslant x) \supset x = y) \ \&$
$((x \leqslant y \ \&\ y \leqslant z) \supset x \leqslant z)]$
[$\leqslant$ is a partial ordering]

$(\forall xh)[(hx \supset \bigcap h \leqslant x) \ \&\ ((\forall y)(hy \supset x \leqslant y) \supset x \leqslant \bigcap h)]$
[every subset h of A possesses a glb $\bigcap h$]

We want to prove :

$(\forall f)[(\forall xy)(x \leqslant y \supset fx \leqslant fy) \supset (\exists z)(fz = z)]$.

The types of our atoms are, respectively :

$x, y, z : \iota$
$h : (\iota \to o)$
$f : (\iota \to \iota)$
$=, \leqslant : (\iota, \iota \to o)$
$\bigcap : ((\iota \to o) \to \iota)$.

We negate the theorem and put all the propositions in clause form. We use again the usual form of Skolem functions, replacing

$E_\iota(\lambda y \cdot hy \supset x \leqslant y)$ by $Y(x,h)$ and
$E_{(\iota \to \iota)}(\lambda f \cdot ((\forall xy)(x \leqslant y \supset fx \leqslant fy) \supset (\exists z)(fz = z)))$
by F , where Y and F are new constants of types respectively $(\iota, (\iota \to o) \to \iota)$ and $(\iota \to \iota)$.

This gives us the following set of clauses :

1 $x \leqslant x$
2 $\neg x \leqslant y, \neg y \leqslant x, x = y$
3 $\neg x \leqslant y, \neg y \leqslant z, x \leqslant z$
4 $\neg hx, \bigcap h \leqslant x$
5 $hY(x,h), x \leqslant \bigcap h$
6 $\neg x \leqslant Y(x,h), x \leqslant \bigcap h$
7 $\neg x \leqslant y, Fx \leqslant Fy$
8 $\neg Fx = x$

First we resolve 2 (on its last literal) and 8. As the constraint $\{x = y, Fx' = x'\}$ possesses a most general unifier $\{x \leftarrow Fx', y \leftarrow x'\}$ we just substitute without generating any constraint, to get :

R(2,8) 9 $\neg Fx \leqslant x, \neg x \leqslant Fx$

Similarly, resolving 4 on its second literal with 9 on its second literal, we get :

R(4,9) 10 $\neg F \bigcap h \leqslant \bigcap h$, $\neg hF \bigcap h$
[The last literal is a shorthand for $\neg(h(F(\bigcap(h))))$].

Next, we resolve the second literal of 7 with the second of 10. Here we do not know a most general way of unifying the two literals, and a constraint is added to the resolvent :

R(7,10) 11 $\neg F \bigcap h \leqslant \bigcap h, \neg x \leqslant y / \{hF \bigcap h,\ Fx \leqslant Fy\}$

(This constraint is obviously unifiable, for instance by $\{h \leftarrow \lambda u \cdot Fx \leqslant Fy\}$).

We shall now resolve both literals of 11 with the second literal of 6. We have to unify :

. firstly $\{F \bigcap h, x, x'\}$

. secondly $\{\bigcap h, y, \bigcap h'\}$ and again we know a most general unifier :

$\{x \leftarrow F \bigcap h, x' \leftarrow F \bigcap h, h' \leftarrow h, y \leftarrow \bigcap h\}$

and therefore we get :

R(6,11) 12 $\neg F \cap h \leqslant Y(F \cap h, h)/\{hF \cap h, FF \cap h \leqslant F \cap h\}$.

Notice that the substitution has been effected in the constraint from 11 as well as in the literals. We shall denote this new constraint by C.
The rest of the refutation is pretty straightforward, no new constraints being generated :

R(3,12) 13 $\neg F \cap h \leqslant y, \neg y \leqslant Y(F \cap h, h)/C$

R(7,13) 14 $\neg \cap h \leqslant y, \neg Fy \leqslant Y(F \cap h, h)/C$

R(4,14) 15 $\neg hy, \neg Fy \leqslant Y(F \cap h, h)/C$

R(5,11) 16 $hY(F \cap h, h)/C$

This last CC is the analogue of 12, 5 and 6 having one literal in common due to the initial reduction to clause form.

Next we resolve both literals of 15 with 16, obtaining therefore a terminal CC :

R(16,15) 17 ϕ/C' where C' consists of the following constraints :

$$\{hF \cap h,\ FF \cap h \leqslant F \cap h\},$$
$$\{h'F \cap h',\ FF \cap h' \leqslant F \cap h'\},$$
$$\{hy,\ Fy \leqslant Y(F \cap h, h),\ h'Y(F \cap h', h')\}.$$

We then apply our unification process to C', which terminates with answer "yes", corresponding to the unifier :

$$\sigma = \{h \leftarrow \lambda u \cdot Fu \leqslant u,\ h' \leftarrow \lambda u \cdot Fu \leqslant u,\ y \leftarrow Y(F \cap \lambda u \cdot Fu \leqslant u,\ \lambda u \cdot Fu \leqslant u)\}.$$

The refutation is now completed, and it is trivial to go back in the proof, substituting σ and the substitutions derived from it by composition with the unifiers already detected, to get a ground refutation.
In particular, we shall substitute $\cap \lambda u \cdot Fu \leqslant u$ for x in 8, which gives us explicitly a fixpoint of F, namely $\cap(\{u \mid Fu \leqslant u\})$, which is precisely its least fixpoint.
Notice that splitting was not necessary here either, and that the refutation is pretty short and does not involve huge constraints.

CONCLUSION

It is difficult to make any efficiency assertion from searchless examples. However, one of the most important heuristic rules to use in this system would be to throw out clauses with constraints which are obviously not unifiable. In practice of course most clauses will be recognized as such, thereby reducing sharply the number of possible derivations.

Types in this logic play a multiple role. First, they permit us to have λ normal forms for all terms. Second, they can be used to sort objects of different nature, and so we get restricted quantification without adding cumbersome additional literals. Finally, as a consequence they restrict the search for unifiers.

This system uses the constraints mechanism to delay as much as possible the process of unification. This has two advantages. First, the deeper we go in a refutation, the more information we gather about the structure of the terms we need to substitute. This reduces our search space for unifiers (intuitively, the more argument-value pairs we know, the less functions exist whose graphs contain these pairs).

Second, we just need to check for the existence of some unifier, we do not need most general ones. This permits us never to consider the unification of two terms whose heads are variables, such as :

$$f(A_1, A_2, \ldots, A_n) \text{ and } g(B_1, B_2, \ldots, B_p)$$
or
$$f(A_1, A_2, \ldots, A_n) \text{ and } f(C_1, C_2, \ldots, C_n).$$

These cases are the most difficult ones to consider for most general unification, but they are trivial when we need only check for existence of unifiers (take any constant function). Actually we need to compose only two elementary unification processes (called imitation and projection), whereas to get most general unifiers Pietrzykowski and Jensen[13] need four, the other two being the most prolific ones indeed. This is because when we consider a pair such as $f(A_1, A_2, \ldots, A_n)$ and $F(B_1, B_2, \ldots, B_p)$, we know something definite about the structure of the common instance (it must begin with an F) and so we have a "handle" on the unification process.

A disadvantage of the delaying of unification is that we may generate clauses whose constraints cannot be satisfied and this is why some processing of these constraints is necessary. In many cases we will know a most general unifier of some constraint and this unifier should be applied to the clause. In other cases we may know that no substitution will unify the constraints, and then we should delete the clause. There is a trade-off here in the amount of unifying while searching which should be done.

Automatic theorem proving is undoubtedly more difficult here than in first-order logic. However our higher-order language permits us to state many theorems in a more concise way, and refutations tend to be a lot shorter. For instance, in our examples, we used set variables in a very natural way (using their characteristic predicates). We could get all the properties of $\in$ with application, of complementation with $\neg$ and of union with $\vee$. Also the abstraction mechanism of λ-calculus replaces in a nice way cumbersome comprehension axioms.

However we might still need use some axioms of descriptions, to assert the existence of certain functions :

$$\exists i\ \forall p[(\exists! x\ p(x)) \supset p(i(p))].$$

When we need to substitute non-atomic propositions for predicate variables (via splitting), refutations tend to become messy and unnatural. The system should be improved on this point, maybe by replacing splitting by rules closer to natural deduction. This would also allow us to get rid of the initial reduction to clause form, which is not always desirable, and to use well-known techniques to decrease the complexity of

refutations (Ernst'). Only then can we hope to deal with proofs by induction in a satisfactory way.

In conclusion, we have presented in this paper a system of logic which is a generalization to Church's type theory of Robinson's resolution. This system seems well suited to mechanizing proofs which require complex substitutions for function variables, as shown in a few hand-simulated examples. Substitutions on non-atomic propositions for predicate variables are not dealt with in a very satisfactory way yet, and some improvement is needed there. Our system without extensionality is equivalent to Andrews' system. However it is amenable to automatic treatment because the substitution rule has disappeared. Our system with extensionality ought to be equivalent to Pietrzykowski's system, and it is conjectured that it is more efficient. A partial implementation is in progress at IRIA, mechanizing constrained resolution without splitting. The emphasis will be put on the heuristic rules for processing constraints during the refutation.

ACKNOWLEDGEMENTS

This paper presents results contained in the author's Ph.D. thesis at Case Western Reserve University, under the direction of Pr. G.W. Ernst. This research was supported by the National Science Foundation under Grant N° GJ-1135.

The author wishes to thank the referees for their helpful comments.

BIBLIOGRAPHY

1 Andrews P.B.(1971)
Resolution in type theory
Journal of Symbolic Logic 36,3 pp.414-432.

2 Andrews P.B.(1972)
General models and extensionality.
Journal of Symbolic Logic 37,2 pp. 395-397.

3 Church A. (1940)
A formulation of the simple theory of types.
Journal of Symbolic Logic 5,1 pp. 56-68.

4 Church A. (1941)
The calculi of lambda-conversion.
Annals of Mathematical Studies no.6.
Princeton University Press.

5 Curry H.B. and Feys R. (1958)
Combinatory Logic (Vol 1)
North Holland, Amsterdam.

6 Ernst G.W.(1971)
A matching procedure for type theory.
Computing and Information Sciences,
Case Western Reserve University.

7 Ernst G.W.(1971)
The utility of independent subgoals in theorem proving.
Information and control 18,3 pp. 237-252.

8 Gould W.E(1966)
A matching procedure for ω-order logic.
Scientific Report N°4, AFCRL 66-781
(Contract AF 19 (628)-3250) AD-646 560.

9 Henkin L.(1950)
Completeness in the theory of types.
Journal of Symbolic Logic 15,2 pp. 81-91.

10 Huet G.P. (1972)
Constrained resolution : A complete method for higher order logic.
Ph.D. Thesis, Case Western Reserve Univ.
Jennings Computing Center Report 1117.

11 Huet G.P. (1973)
The undecidability of unification in third order logic.
Information and Control 22,3 pp. 257-267.

12 Pietrzykowski T.(1971)
A complete mechanization of second order logic.
Report CSRR 2038, Department of Applied Analysis and Computer Science,
University of Waterloo.
Also Journal of Assoc. for Comp. Mach. 20,2 pp. 333-364.

13 Pietrzykowski T. and Jensen D. (1972)
A complete mechanization of ω-order type theory.
Report CSRR 2060, Department of Applied Analysis and Computer Science,
University of Waterloo.
Also Assoc. for Comp. Mach. National Conf. (1972). Vol 1, pp. 82-92.

14 Robinson J.A.(1965)
A machine-oriented logic based on the resolution principle.
Journal of Assoc. for Comp. Mach. 12,1 pp. 23-41.

15 Robinson J.A.(1968)
New directions in theorem proving.
Proceedings of IFIP Congress Edinburgh August 1968 in Information processing 68, Vol 1 pp. 63-67, North-Holland, Amsterdam.

16 Robinson J.A. (1969).
Mechanizing higher order logic.
Machine Intelligence 4, pp. 151-170.
American Elsevier, New York.