

SUB-PROBLEM FINDER AND INSTANCE CHECKER
TWO COOPERATING PREPROCESSORS FOR THEOREM PROVERS

Dennis de Champeaux
Bedrijfsinformatica
University of Amsterdam
Jodenbreestraat 23 K3123
1011 NH Amsterdam

Two pre-processors for theorem provers are described, which when applicable, will lead to search space simplification.

Both were implemented and integrated with an existing resolution type connection graph theorem prover.

Examples are provided which confirm our claim of search space simplification.

Key words and phrases: Theorem proving, pre-processing, search space simplification.

1. Pre-processors for theorem provers in general.

The main paradigm in automatic theorem proving is or should be that search is to be avoided, postponed, or else to be minimized. This should be done by any means one can lay one's hand on while maintaining completeness, generality and not succumbing to the ad-hocness as advertised by the proceduralists.

Search space simplification has been a major goal in the resolution school. Exploiting 'larger' more numerous operators (derivation rules) is the main activity in the natural deduction school. There are however many more options to be developed with which the role of search can be limited to those circumstances where there is no sensible alternative.

Already slumbering for some years is the application of multi-level search in theorem proving (Sacerdoti, 1973 [7]). It is incomprehensible that the support from a model technique (Gelertner, 1959 [3]), has not been pursued. Search guidance with heuristic functions and automatic improvement of them is still inconclusive, although it has become clear that syntactic features can contribute only modestly to such functions. Choosing an appropriate representation (within one language say predicate calculus) is a wide open problem. Determining the right representation language is not yet a problem since it is not known whether multiple representation languages are necessary.

The former two issues belong to the preprocessing repertoire. We mention a few more. Selection of relevant axioms and/or definitions and/or already

proven theorem/lemmas to prove a conjecture; finding a counter example of a conjecture to make sure that a proof is impossible; finding a similar already proven theorem to see whether its proof can be generalized and/or modified to handle a conjecture; instantiating a second order theorem (e.g. induction scheme); reducing a conjecture to independent sub-problems (reduction to weakly dependent sub-problems can be done in the multi-level search framework); recognizing that a conjecture is an alphabetic variant and/or an instantiation of an axiom or an already proven theorem; etc.

This paper reports the results of implementing the two last mentioned. The next section describes the independent sub-problem recognizer. Section 3 deals with the instance checker. Section 4 discusses how they are cooperating and how they should be related from a process point of view. Two examples are given in section 5.

2. The independent sub-problem recognizer.

The sub-problem recognizer we developed is based on components of a predicate calculus-conjunctive normal form translator. Our translator was inspired by the procedure as described in [Manna, 6]. A small improvement to this translator led to sub-problem decomposition. First we present the original translation. Then we expand one of the steps (as was also partially done in [Loveland, 5] page 34). A subset of these translator rules make up the sub-problem recognizer.

The translator in [Manna, 6] omitting here irrelevancies, consists of the following steps:

1- Eliminate 'if ... then' and 'if and only if'.

Replace $A \supset B$ by $V(\neg A, B)$, and
 $A \equiv B$ by $\Lambda(V(\neg A, B), V(A, \neg B))$.

2- Move 'not' inwards.

Replace $\neg(x)A$ by $(\exists x)\neg A$,
 $\neg(\exists x)A$ by $(x)\neg A$,
 $\neg V(A_1, \dots, A_n)$ by $\Lambda(\neg A_1, \dots, \neg A_n)$,
 $\neg \Lambda(A_1, \dots, A_n)$ by $V(\neg A_1, \dots, \neg A_n)$, and
 $\neg A$ by A .

3- Push quantifiers to the right.

Let (Qx) be (x) or $(\exists x)$, and let XX be Λ or V ;
 replace $(Qx) XX(A_1, \dots, A_i, \dots, A_n)$ by
 $XX(A_i, (Qx)XX(A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_n))$
 if x not free in A_i .

4- Eliminate existential quantifiers (introduction of Skolem functions). Pickout the leftmost well formed part $(\exists y)(B(y))$ and replace it by

$B(f(x_{i1}, \dots, x_{in}))$ where

a) $x_{i1}, \dots, x_{in}$ are the free variables of
 $(\exists y)(B(y))$ which are universally quantified
 to the left of $(\exists y)(B(y))$;

b) f is a 'fresh' n -ary function constant.

5- Eliminate universal quantifiers.

6- Distribute 'and' over 'or'.

Replace $V(A_1, \dots, A_i, \Lambda(B_1, \dots, B_k),$
 $A_{i+1}, \dots, A_n)$ by
 $\Lambda(V(A_1, \dots, B_1, \dots, A_n),$
 $\dots, V(A_1, \dots, B_k, \dots, A_n))$.

If step (i) can be reapplied it has precedence over step (i+1).

Remark: A sequence of say universal quantifiers should get special attention in step 3. E.g.

$(x)(y)\{VP(x, y), Q(y)\}$ can be replaced by
 $(y)V\{Q(y), (x)P(x, y)\}$.

Thus every universal (existential) quantifier in a sequence of universal (existential) quantifiers should be moved to the right of the sequence in its turn to check whether it can be pushed further to the right.

This procedure can be improved by expanding step 3 with:

3.1 Straighten out 'and's and 'or's.

Let XX be Λ or V .

Replace $XX(A_1, \dots, A_i, XX(B_1, \dots, B_k),$
 $A_{i+1}, \dots, A_n)$ by
 $XX(A_1, \dots, A_i, B_1, \dots, B_k,$
 $A_{i+1}, \dots, A_n)$.

3.2 Distribute 'and' over 'or' or vice versa.

Replace
 $(x)V(A_1, \dots, A_i, \Lambda(B_1, \dots, B_k),$
 $A_{i+1}, \dots, A_n)$ by
 $(x)\Lambda(V(A_1, \dots, B_1, \dots, A_n),$
 $\dots, V(A_1, \dots, B_k, \dots, A_n))$ and
 $(\exists x)\Lambda(A_1, \dots, A_i, V(B_1, \dots, B_k),$
 $A_{i+1}, \dots, A_n)$ by
 $(\exists x)V(\Lambda(A_1, \dots, B_1, \dots, A_n),$
 $\dots, \Lambda(A_1, \dots, B_k, \dots, A_n))$.

3.3 Distribute quantifiers over connectives.

Replace

$(x)\Lambda(A_1, \dots, A_n)$ by $\Lambda((x)A_1, \dots, (x)A_n)$ and
 $(\exists x)V(A_1, \dots, A_n)$ by $V((\exists x)A_1, \dots, (\exists x)A_n)$.

Example: Using the extended step 3 the formula:

$(x)[P(x) \wedge (y)\{Q(x, y) \wedge (\exists z)R(y, z)\}]$
 can be rewritten into:
 $(x)P(x) \wedge (y)(x)Q(x, y) \wedge (y)(\exists z)R(y, z)$.

Observe that this replacement simplifies the Skolem function that has to be generated for the elimination of the $(\exists z)$ -quantifier.

Remark: It is worthwhile to check for redundancies (using the instance checker) after application of step 3.2.

E.g. $V((x)A(x), A(a), \dots)$ can be simplified to
 $V(A(a), \dots)$ while
 $\Lambda((x)A(x), A(a), \dots)$ can be replaced by
 $\Lambda((x)A(x), \dots)$.

Remark: For easy readability we used in 3.3 the same variable 'x' in all the terms of the connective in the replacement. This does not hurt the translator. The instance checker however can get confused and demands step 3.3 to be modified such that each term in the replacement has a new fresh variable.

The INdependent SUB-problem REcognizeR - INSURER - consists of step 1 upto the modified step 3 and with step 6. It is obvious that in case INSURER applied on a problem P produces a form with leading connective 'and' its terms are independent sub-problems and if all sub-problems can be proven the P has been solved. The reverse statement that INSURER will find a maximal decomposition if P can be decomposed we leave as an intriguing task for Aiello/Weyrauch's program FOL [1].

We end this section with the 'real-life' example given in box 1.

(1)	$(x)(y)(z) x(yz)=(xy)z$
(2)	$(x) xe=x$
(3)	$(x) ex=x$
(4)	$(x) xI(x)=e$
(5)	$(x) I(x)x=e$
(6)	$(H) (SUBGR(H) \leftrightarrow \{ (\exists x)H(x) \wedge$ $(x)(y) [H(x) \wedge H(y) \rightarrow H(xy)] \wedge$ $(x)(H(x) \rightarrow H(I(x))) \})$
(7)	$(H1)(H2) \{ SETEQ(H1, H2) \leftrightarrow (x)(H1(x) \leftrightarrow H2(x)) \}$
(8)	$(g)(xx)(H) (COSET(g, xx, H) \leftrightarrow$ $\{ SUBGR(H) \wedge$ $(x)(xx(x) \leftrightarrow (\exists y)(H(y) \wedge xyg)) \})$
(9)	$(g)(xx)(H) (COSET(g, xx, H) \rightarrow \{ H(g) \leftrightarrow SETEQ(xx, H) \})$.

box 1 Axioms (1-5), definitions (6-8) and a theorem (9) from group theory.

(1-5) are non-minimal axioms defining a group; (6-8) are definitions of respectively sub-groups, equality of subsets and of right-cosets; (9) is a theorem expressing a property of cosets. Observe that subsets are represented by 1-argument first order predicates, and that SETEQ and COSET are 2nd order predicates. Direct translation of (1-8) and the negation of (9) into conjunctive normal form yields 39 clauses with together 109 literals. INSURER however recognizes that (9) can be decomposed into:

- (10) $(g)(H)\{H(g)V(xx)[\neg \text{COSET}(g,xx,H)V \neg \text{SETEQ}(xx,H)]\}$ and
 (11) $(g)(H)\{\neg H(g)V(xx)[\neg \text{COSET}(g,xx,H)V \neg \text{SETEQ}(xx,H)]\}$.

Working on (10) (not done by INSURER, but by another program component) the definition of COSET, SETEQ and SUBGR are respectively substituted. The result is negated and together with (1-5) translated into conjunctive normal form yielding 14 clauses with 23 literals. After each substitution of a definition INSURER is called to check whether further decomposition is possible. When working on (11) this strategy is successful after substituting away SETEQ. Two new sub-problems are found both ending up with 14 clauses and 21 literals.

Although our connection graph theorem prover COGITO is not yet able to handle these three sub-problems, the chance to find a solution has increased with an 'infinite' amount when compared to the non-decomposed situation.

INSURER also can handle the sorted predicate calculus that was introduced in [Champeaux, 2]. The same coset example formulated in sorted predicate calculus - without decomposition - yields 28 clauses with 61 literals. INSURER finds here also three sub-problems each having 12 clauses with respectively 16, 14 and 14 literals. A significant reduction again, possibly bringing this problem within reach of the with paramodulation extended COGITO.

3. The instance checker

The instance checker (INSTANCE) we designed and implemented is in fact a special case theorem prover. It is an instrument with which a conjecture can be recognized as being an alphabetic variant or as a special case of an already accepted theorem. Conjecture and theorem are expressed as closed, slightly restricted - see below - predicate calculus formulas.

Let T and K be respectively a conjecture and an accepted piece of 'knowledge'. The input of the

recursive INSTANCE consists of three elements:
 -- two forms T and K for which must hold that they do not share variables and that dis-regarding permutations of sequences of quantifiers and/or arguments of 'and' and 'or' it is the case that $T = \text{INSURER}(T)$ and $K = \text{INSURER}(K)$; and
 -- a list of variables, VR, to be explained in the sequel, which at the top level call is empty.
 Although at the top level one is mostly interested in a yes-no answer, for reasons that should become clear in the description of INSTANCE, the output is more substantial in the positive case. The output of INSTANCE is:
 -- NO, signifying that T is not an instance of K, or else
 -- a list of triples, where each triple is of the form $\{\sigma, T\sigma, K\sigma\}$ with σ a substitution, T a "without loss of generality substitution instance" of T being a logical instance of the special case $K\sigma$ of K. It should now be clear that if $\text{NO} \neq \text{INSTANCE}(T, K, \emptyset)$ then $K \vdash T$.

An example where more than one triple will be returned by INSTANCE is:

$T = (\exists x)Ax, x$,
 $K = (y)(A(p, y) \wedge A(q, y))$, with the output:
 $(\{ \begin{smallmatrix} x \leftarrow p \\ y \leftarrow p \end{smallmatrix}, A(p, p), A(p, p) \wedge A(q, p) \})$
 $(\{ \begin{smallmatrix} x \leftarrow q \\ y \leftarrow q \end{smallmatrix}, A(q, q), A(p, q) \wedge A(q, q) \})$.

We give a simplified description of INSTANCE omitting subtleties that have to do with the equality predicate. α stands for: the identifier α occurs in the expression y ; S_{β}^{α} is the result of substituting α for β in the expression y ; thus $S_{\beta}^{\alpha} = y\sigma$ when $\sigma = \{\beta \leftarrow \alpha\}$.

We assume that the employed unification algorithm is willing to accept also non-literal predicate calculus formulas and so we get for example $\{z \leftarrow a\}$ when calling $\text{UNIFY}((x)A(x, a), (x)A(x, z), \{z\})$.
 $\text{INSTANCE}(T, K, \text{VR}) :=$
 if T and K are unifiable, taking into account VR, under substitution σ
 then return with $(\{\sigma, T\sigma, K\sigma\})$
 else
 if $K = (\alpha)(\text{Form}(\alpha))$
 then $U := \text{INSTANCE}(T, \text{Form}(\alpha), \text{VRU}\{\alpha\})$
 if $U = \text{NO}$ then return with NO
 else $(\text{thus } U = \bigcup_{i=1}^n \{\sigma_i, x_i, y_i\})$
 return $\bigcup_{i=1}^n \{\sigma_i, x_i, \tilde{y}_i\}$,
 where $\tilde{y}_i = (\alpha)y_i$ when $\alpha \in y_i$ or else y_i
 else
 if $T = (\exists \alpha)(\text{Form}(\alpha))$
 then $U := \text{INSTANCE}(\text{Form}(\alpha), K, \text{VRU}\{\alpha\})$

```

    if U=NO then return with NO
    else return  $\bigcup_1 \{\sigma_i, \tilde{x}_i, y_i\}$ 
      where  $\tilde{x}_i = (\exists \alpha) x_i$  when  $\alpha \in x_i$  or else  $x_i$ 
  else
    if T =  $(\alpha)(\text{Form}(\alpha))$ 
      then let c be a fresh constant
      U := INSTANCE( $S_\alpha^C(\text{Form}(\alpha), K, \text{VR})$ )
      if U=NO then return with NO
      else return  $\bigcup_1 \{\sigma_i, \tilde{x}_i, y_i\}$ 
        where  $x_i = (\alpha) S_\alpha^C x_i$  when  $c \notin y_i$ 
        or else  $x_i$ 
    else
      if K =  $(\exists \alpha)(\text{Form}(\alpha))$ 
        then let c be a fresh constant
        U := INSTANCE(T,  $S_\alpha^C \text{Form}(\alpha), \text{VR}$ )
        if U=NO then return with NO
        else return  $\bigcup_1 \{\sigma_i, x_i, \tilde{y}_i\}$ 
          where  $\tilde{y}_i = (\exists \alpha) S_\alpha^C y_i$ 
          when  $c \notin x_i$  or else  $y_i$ 
      else
        if K =  $V(K_1, \dots, K_n)$ 
          then return  $\bigcup_1 \{\sigma_i, x_i, y_i\}$  where  $x_i = T\sigma_i, y_i = K\sigma_i$ 
          and for all j  $x_j$  is an instance of  $K_j\sigma_j$  or if
          no such triple exists NO [see the appendix
          for a more detailed description of this case]
        else
          if T =  $\Lambda(T_1, \dots, T_n)$ 
            then return  $\bigcup_1 \{\sigma_i, x_i, y_i\}$  where  $x_i = T\sigma_i, y_i = K\sigma_i$ 
            and for all j  $T_j\sigma_j$  is an instance
            of  $y_j$  or if no such triple exists NO [we omit
            specification of this case since it runs
            parallel to the former case]
          else
            if T =  $V(T_1, \dots, T_n)$ 
              then U :=  $\emptyset$ 
              for all  $T_j$  do
                U2 := INSTANCE( $T_j, K, \text{VR}$ )
                if U2  $\neq$  NO then  $\bigcup_1 \{\sigma_i, x_i, y_i\}$  then
                  U :=  $\bigcup_1 \{\sigma_i, T\sigma_i, y_i\}$ 
                if U =  $\emptyset$  then return NO else return U
            else
              if K =  $\Lambda(K_1, \dots, K_n)$ 
                then U :=  $\emptyset$ 
                for all  $K_j$  do
                  U2 := INSTANCE(T,  $K_j, \text{VR}$ )
                  if U2  $\neq$  NO then U :=  $\bigcup_1 \{\sigma_i, x_i, K\sigma_i\}$ 
                else return NO.

```

As mentioned in the former section INSTANCE is called in INSURER after application of step 3.2. The effectiveness of doing so was proven by the conjecture of the second example in section 5: $(x)(\exists y)(\wedge(\text{Equal}(x, \text{nil}) \dots$ Without INSTANCE INSURER will decompose this example into eight (8!) sub-problems of which six are redundant. When INSTANCE is incorporated the two non-redundant sub-problems only remain. INSURER and INSTANCE can also be coupled in another way as we will describe in the next section

4. Interplay between INSURER, INSTANCE and COGITO and what is to be desired.

INSURER, INSTANCE, COGITO and the predicate calculus - conjunctive normal form translator - were embedded in a 'fixed' regime. Input for the prover consists of axioms, supporting theorems, definitions and the conjecture. For the next description we want to remind that activation of the connection graph theorem prover COGITO should be postponed at all costs.

Roughly a supervisor triggers the following activities:

- step 1: If the conjecture is an instance of an axiom, a theorem or an already proven theorem (see step 2) then return with success.
- step 2: If the conjecture decomposes into the sub-problems $C_1, \dots, C_n$ then for each C_i go (recursively) to step 1 if the value returned for treating C_i is succesful then add C_i to the collection of already proven theorems else quit with failure return with success.
- step 3: If the conjecture contains a predicate defined in one of the definitions then substitute for each occurrence in the conjecture the instantiated body of the definition* and go to step 1.
- step 4: Translate the axioms, supporting theorems and the negation of the conjecture into conjunctive normal form, call COGITO and return the value returned by it. (COGITO gets a resource parameter ensuring termination).

The results reported in the next section have been obtained with this, slightly different, regime. Although this particular fixed connection between INSURER, INSTANCE and COGITO makes sense and is certainly effective we do not like it. We prefer considering INSURER, INSTANCE and COGITO as being members of a potentially larger family of deductive, cooperating, independent specialists. This would require the supervisor to be implemented as a multi-processes scheduler. The overall structure would be more transparant, making more easily addition of a new specialist. Not having available language as QLISP, INTERLISP and MAGMA-LISP prevented us of doing so.

5. Examples

Our first example looks terribly simple but a straightforward treatment after translation, by COGITO had not yet found a contradiction after

* Recursive definitions are not allowed

generating 35 clauses. It consists out of only:
definition: $(s)(t)\{SETEQ(s,t) \leftrightarrow (x)(x \in s \leftrightarrow x \in t)\}$
and conjecture: $(u)(v)\{SETEQ(u,v) \leftrightarrow SETEQ(v,u)\}$.
INSURER immediately recognizes that the conjecture decomposes into two sub-problems of which INSTANCE shows that one is an alphabetic variant of the other.

Remains to be proven:

$(u)(v)\{\neg SETEQ(u,v) \vee SETEQ(v,u)\}$.

After substituting away SETEQ with the definition INSURER finds again two sub-problems and - surprise - INSTANCE also finds that one is a variant of the other.

Remains this time:

$(u)(v)\{V(\exists y1)(y1 \in v \wedge y1 \notin u)$
 $(\exists y2)(y2 \notin v \wedge y2 \in u)$
 $(y3)(y3 \in v \vee y3 \notin u)\}$.

Negating this conjecture and translating into C.N.F. yields four clauses. The 2nd clause generated by COGITO was already the empty one. Generating the non-solution with 35 clauses was 20 times more costly than finding this solution.

The next example was taken from [Green, 4] and was already worked on as reported in [Champeaux, 2].
definition:

$(x)(y)\{R(x,y) \leftrightarrow (Same(x,y) \wedge Sd(y))\}$.

axioms:

(1) $(x)(y)\{Sd(y) \rightarrow Sd(merge(x,y))\}$,
(2) $(x)(y)(u)\{(Sd(y) \wedge Same(x,y)) \rightarrow$
 $\rightarrow Same(cons(u,x), merge(u,y))\}$,
(3) $(x)\{Equal(x,nil) \rightarrow R(x,nil)\}$,
(4) $(x)\{\neg Equal(x,nil) \rightarrow$
 $\rightarrow Equal(x, cons(car(x), cdr(x)))\}$,

(5) $(x)(u)(v)\{Equal(x,u) \wedge Same(u,v) \rightarrow Same(x,v)\}$.

conjecture:

$(x)(\exists y)\{(Equal(x,nil) \rightarrow R(x,y)) \wedge$
 $((\neg Equal(x,nil)) \wedge$
 $R(cdr(x), sort(cdr(x))) \rightarrow$
 $\rightarrow R(x,y))\}$.

The conjecture - already mentioned in section 3 - decomposes into two sub-problems of which INSTANCE finds that one is an instance of axiom(3). The remaining sub-problem was solved as well as without definition substitution (by adding the definition to the axioms) as with substitution. In both cases a contradiction was found more easily than in the non-decomposed case as can be seen from the g-penetrance values in table 1.

program and strategy	input + generated clauses	g-penetrance
QA3 [Green, 4]	286	0.091
COGITO (plain)	38 (25)	0.579 (0.680)
+ sub-problem recognition	28 (17)	0.785 (0.882)
+ definition substitution	20 (12)	0.800 (0.917)

Table 1 Showing the effectiveness of INSURER and INSTANCE. The numbers between brackets refer to values obtained when the sorted predicate calculus is used [Champeaux, 2]. (The g-penetrance is defined as $\frac{\# \text{ clauses in proof}}{\# (\text{input} + \text{generated clauses})}$.)

6. Summary

Two algorithmic (always halting) components from the natural deduction realm are described. Both were implemented and integrated as pre-processors with a resolution type, connection graph theorem prover. Examples are given that illustrate augmented power of this complex with respect to the sole theorem prover.

Factoring out other algorithmic components and/or adding plan/strategy generators to the pre-processor family (possibly leading to multi-level search [Sacerdoti, 7], we consider a more promising future task than investing more intelligence deep inside resolution and/or natural deduction theorem provers.

Excellent typing was done by Pom van der Horst

Appendix

We give here a more detailed description of the sub-case

INSTANCE(T, V(K1, ..., Kn), VR).
INSTANCE(T, V(K1, ..., Kn), VR):=
INSORK(({K1, ..., Kn}(\emptyset, T, \emptyset)))

So we have now to describe the procedure INSORK. Its input consists of a non-empty list (Task1, ..., Taski, ...), where an element Taski is of the form:

$\{(\tilde{R}_m, \dots, \tilde{R}_n)(\sigma, \tilde{T}, (\tilde{R}_1, \dots, \tilde{R}_{m-1}))\}$, with $\tilde{R}_j = K_j \sigma$, $\tilde{T} = T \sigma$. [$\tilde{T}$ is already an instance with respect to $V(\tilde{R}_1, \dots, \tilde{R}_{m-1})$]

INSORK($\bigcup_i$ Taski):=
Newtask:= $\emptyset$
for each Taski do
U:=INSTANCE($\tilde{T}$, $\tilde{R}_m$, VR)
if U $\neq$ NO thus U=U $\{\sigma_j, x_j, y_j\}$
then for each element of U do
NN:=($\tilde{R}_m + \sigma_j, \dots, \tilde{R}_n \sigma_j$)
 $(\sigma + \sigma_j, x_j, (\tilde{R}_1 \sigma_j, \dots, \tilde{R}_{m-1} \sigma_j, y_j))$
Newtask:=NewtaskUNN
if Newtask= $\emptyset$ then return NO
if m=n thus all elements Ki have been treated
then U2:= $\emptyset$
for each element of Newtask do
(which have the form
 $\{\emptyset, (\sigma, \tilde{T}, (\tilde{R}_1, \dots, \tilde{R}_n))\}$)
U2:=U2U $\{\sigma, \tilde{T}, V(\tilde{R}_1, \dots, \tilde{R}_n)\}$
return U2
else return INSORK(Newtask).

REFERENCES

- 1 Aiello, M. & Weyrauch, R.W., Checking Proofs in the Metamathematics of First Order Logic, Stanford A.I.Lab.Memo AIM-22, August 1974.
- 2 Champeaux, D. A theorem prover dating a semantic network, Proceedings of the AISB/GI Conference on A.I., pp. 82-92, Hamburg 1978
- 3 Gelernter, H. et al, Empirical Explorations of the Geometry Theorem Proving Machine, reprinted in: Computers and Thought, Ed. Feigenbaum-Feldman, McGraw-Hill, 1963, pp. 153-163.
- 4 Green, C., The application of Theorem Proving to Question-answering Systems, AI Group, Technical Note 8, SRI, Stanford 1969.
- 5 Loveland, D.W., Automated Theorem Proving: A Logical Basis, North-Holland, 1978.
- 6 Manna, Z., Introduction to Mathematical Theory of Computation, McGraw-Hill, 1972.
- 7 Sacerdoti, E., Planning in a Hierarchy of Abstraction Spaces, IJCAI3, pp. 412-422, Stanford, 1973.

P.S.

W. Bibel brought under our attention the paper Ernst, G.W., The Utility of Independent Subgoals in Theorem Proving, Information and Control 18, pp. 237-252, 1971. A two level process is described where the first level corresponds with the sub-problem finder INSURER. The differences are:
 - rule 3.2 is missing; - the check for redundancy using INSTANCE inside INSURER is missing.

Recently we have extended INSURER with additional INSTANCE-checking such that the role of INSTANCE can be formulated as:

Let $X = \Lambda(A, A_1, \dots, A_i, \dots, A_n)$;

if INSTANCE(A, A₁)

then replace X by $\Lambda(A_1, \dots, A_i, \dots, A_n)$;

if INSTANCE(movenot($\neg A$), A_i) then replace X by false;

and similar rules for disjunctions. (See Quine, W.O., Methods of Logic, Holt NY, 1950, pp. 101-107, which mentions special cases.)

INSURER thus fortified was capable to reduce to true:

$$\begin{aligned} & \boxed{\exists x_1 (y_1) P(x_1) \equiv P(y_1)} \equiv \boxed{\exists x_2 Q(x_2) \equiv (y_2) P(y_2)} \equiv \\ & \boxed{\exists x_3 (y_3) Q(x_2) \equiv Q(y_3)} \equiv \boxed{\exists x_4 P(x_4) \equiv (y_4) Q(y_4)} \end{aligned}$$

(This problem was posed by P. Andrews at the Fourth Workshop on Automated Deduction, Austin, Feb. 1979.)

Question: Can INSURER - with these INSTANCE rewrite rules - recognize every valid formula from the monadic predicate calculus?