Subproblem Finder and Instance Checker, Two Cooperating Modules for Theorem Provers

DENNIS DE CHAMPEAUX

University of Amsterdam, The Netherlands

Abstract. Properties are proved about INSTANCE, a theorem prover module that recognizes that a formula is a special case and/or an alphabetic variant of another formula, and about INSURER, another theorem prover module that decomposes a problem, represented by a formula, into independent subproblems, using a conjunction. The main result of INSTANCE is soundness; the main result of INSURER is a maximum decomposition into subproblems (with some provisos). Experimental results show that a connection graph theorem prover extended with these modules is more effective than the resolution-based connection graph theorem prover alone.

Categories and Subject Descriptors: I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*deduction*

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Alphabetic variant recognition, meta properties, problem decomposition, special case recognition

1. *Introduction*

Deduction, although a proper subset of problem solving, includes a vast range of activities. This spectrum extends from confirming routine conjectures—for instance, type checking in low-level natural language processing—to proving theorems in mathematics.

It is improbable that one and the same vehicle could control the whole deductive area. The trade-off between generality and specialization may be employed to use different data structures or languages for different tasks. For instance, deduction for routine conjecture verification may be reformulated as pointer chasing in a tree hierarchy (under certain circumstances). At another extreme, deduction in a mathematical context may be seen as "obtaining the right point of view" or "getting the right kind of formalization," which hides irrelevancies until, at the right moment, sufficient similarity with previously solved theorems has been attained.

Reformulating, in the latter case, will not, in general, be enough to solve a new problem. An operator that applies in a formerly established proof cannot be applied immediately to the problem at hand; additional cases must be considered, and so forth. These gaps may lead to the generation of new conjectures, to be handled in the same high-level spirit; else, unavoidably, it may be necessary to rely on brute force search.

Author's present address: 14519 Bercaw Lane, San Jose, CA 95124.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM 0004-5411/86/1000-0633 $00.75

Classical theorem proving, like resolution and natural deduction, is, in our opinion, an excellent tool to be invoked when high-level reasoning cannot be applied any longer and it becomes necessary to rely on blind search, although it should be used as cleverly as possible to prevent, for example, two commuting resolution operations. However, it is our opinion, as well, that application of these techniques should be postponed at all costs. The problem is that despite great progress in resolution and natural deduction, it remains virtually impossible to make these search techniques "aware" of nonsyntactical features.

These techniques can be criticized for not being sensitive to "obvious" peculiarities of problems to be solved. They do not "know" the difference between an axiom, a theorem, a definition, or a recursive definition. They are not flexible about deciding whether to try proving a subgoal or to try finding a counterexample instead. They cannot juggle several interpretations of a set of formulas in order to guide decisions. They cannot recognize that a proof, in fact, allows the assertion of a stronger conclusion than the conjecture started off with. (When an assumption is introduced and it can be recognized that the subsequent proof does not depend on it, one can prevent repeating the proof under the negated assumption.) They are ignorant of other theories and thus cannot attempt to adapt proofs by analogical reasoning (with the notable exception of a first attempt in [9]).

In summary, the tool box of deduction rules is supposed to do too much work, and its tools are clumsy, while useful data—models, other theories, distinctions between formulas or, even the proof sequence of already proven theorems—are inaccessible.

Consequently, we envision a different "architecture" for more powerful theorem provers. They should be arranged as cooperating deductive *specialists*, each one embodying sound deductive power and, when productive—in the sense of a production system—able to ensure a positive contribution to a solution. No single one needs to be complete. When, for example, a resolution specialist belongs to the community, completeness for the community is ensured.

Of course another problem, well known in production systems circles, arises when there are many deductive specialists available: How does one decide cheaply which specialist is applicable and, when at least two are applicable, which should be given control? When we become accustomed to supplying more information than just axioms and a conjecture to a community of deductive specialists, as argued above, this problem might be alleviated in a non-ad-hoc way.

We describe in this paper two deductive specialists:

(1) INSTANCE. INSTANCE is able to decide whether a conjecture, represented by a formula, is a special case, an alphabetic variant, and/or an and/or-connective permutation variant of an already accepted axiom, theorem, lemma, intermediate result, and so forth.

(2) INSURER. INSURER is able to recognize when a conjecture, represented by a formula, can be rewritten into independent, easier-to-handle subproblems, using a conjunction.

These specialists—which, alternatively, can be seen as preprocessors for a conventional theorem prover—together with an unsophisticated applier of definitions, have been implemented and integrated with a connection graph resolution-based theorem prover. The augmented power of this deduction complex with respect to the sole theorem prover is shown by examples.

Some of this work has been reported in [2]. Meanwhile, the cooperation between the two preprocessors could be increased, making one of them still more effective and maintaining their algorithmic, always halting nature. The cooperation has become so close that we have the following apparent paradox: Although the output format of INSURER is the input format of INSTANCE, INSTANCE has processing responsibilities deep down inside INSURER.

Section 2 is devoted to a tutorial example showing how INSTANCE and INSURER take care of obvious features (without which our regular resolution theorem prover flounders). The quick reader can safely skip to the end after this section. The next section is devoted to the definition of INSTANCE and a description of some of its properties. In Section 4, INSURER is defined and its properties are explored. Section 5 describes the structure of the supervisor of the theorem prover COGITO. This supervisor contains INSTANCE, INSURER, a definition opener, and a connection graph resolution component, which handle the examples to be discussed in Section 6. Related work is compared in Section 7.

2. *A Tutorial Example*

Our theorem prover consists of several relatively independent modules. In addition to INSTANCE and INSURER, there are a simple-minded applier of definitions, a translator from Predicate Calculus (PC) to conjunctive normal form (CNF), a connection graph resolution component, and an unsophisticated supervisor.

The following example from elementary set theory looks trivial, but a straightforward treatment by our resolution component, after translation into CNF, has not yet found a contradiction after generating 35 clauses. It consists only of the following:

Definition

$$(s)(t)\{\text{SETEQ}(s, t) \leftrightarrow (x)\ (\text{IN-SET}(x, s) \leftrightarrow \text{IN-SET}(x, t))\}.$$

Conjecture

$$(u)(v)\{\text{SETEQ}(u, v) \leftrightarrow \text{SETEQ}(v, u)\}.$$

The subproblem finder INSURER immediately recognizes that the conjecture reduces to

$$(u)(v)\{\sim\text{SETEQ}(u, v) \vee \text{SETEQ}(v, u)\} \wedge$$
$$(u)(v)\{\text{SETEQ}(u, v) \vee \sim\text{SETEQ}(v, u)\}.$$

The alphabetic variant/variable(s) instantiation checker INSTANCE will recognize that the second subproblem is an alphabetic variant of the first subproblem. Thus the deduction complex has only to deal with the first one. The module specializing in "opening" defined predicates will recognize that the complex can be applied and will rewrite the first subproblem into

$$(u)(v)\{\sim(x)(\text{IN-SET}(x, u) \leftrightarrow \text{IN-SET}(x, v)) \vee$$
$$(x)(\text{IN-SET}(x, v) \leftrightarrow \text{IN-SET}(x, u))\}.$$

Again INSURER will be invoked for this formula. To appreciate its result, we focus more closely on its actions. First $\leftrightarrow$ is removed and $\sim$ is moved inward,

resulting in

$$\begin{aligned}(u)(v)[\vee((E\ x)\{&\text{IN-SET}(x, u) \wedge \sim\text{IN-SET}(x, v)\}\\ (E\ x)\{&\sim\text{IN-SET}(x, u) \wedge \text{IN-SET}(x, v)\}\\ (\wedge(x)\{&\sim\text{IN-SET}(x, v) \vee \text{IN-SET}(x, u)\}\\ (x)\{&\text{IN-SET}(x, v) \vee \sim\text{IN-SET}(x, u)\}\})].\end{aligned}$$

The structure of this formula is

$$\begin{aligned}(u)(v)[\vee\ &O_1\\ &O_2\\ &(\wedge\ A1 \quad A2)].\end{aligned}$$

Since the body of the quantifiers is a disjunction, the universal quantifiers cannot be distributed. The third argument of the disjunction is a conjunction, however, allowing the production of a conjunction as the body of the quantifiers. This produces

$$\begin{aligned}(u)(v)[\wedge\ &\{\vee\ A_1 \quad O_1 \quad O_2\}\\ &\{\vee\ A_2 \quad O_1 \quad O_2\}].\end{aligned}$$

Before pushing the quantifiers to the right, the disjunctions are scrutinized, since simplifications may be possible after the distribution. The first disjunction, $\{\vee\ A_1\ O_1\ O_2\}$, is in fact

$$\begin{aligned}[\vee\ &(x)\{\sim\text{IN-SET}(x, v) \vee \text{IN-SET}(x, u)\}\\ &(E\ x)\{\text{IN-SET}(x, u) \wedge \sim\text{IN-SET}(x, v)\}\\ &(E\ x)\{\sim\text{IN-SET}(x, u) \wedge \text{IN-SET}(x, v)\}].\end{aligned}$$

INSTANCE will observe that the negation of the first argument $\sim A_1$, after working the negation inwards producing

$$(E\ x)\{\text{IN-SET}(x, v) \wedge \sim\text{IN-SET}(x, u)\},$$

is the same formula as the third argument O_2. Thus, this disjunction can be collapsed to TRUE. This result of INSTANCE does not depend on the variable x being used in A_1 as well as in O_2. INSTANCE would have also reported success when O_2 was, for example,

$$(E\ y)\{\sim\text{IN-SET}(y, u) \wedge \text{IN-SET}(y, v)\}.$$

The second disjunction, $\{\vee\ A_2 \quad O_1 \quad O_2\}$, collapses in a similar way. Thus, the whole formula rewrites into TRUE, and we are finished. Notice that the resolution component was not called upon.

3. *Compressed MiniScope and INSTANCE*

As discussed in the former section, distinct deductive specialists may require distinct data representations for the objects on which they operate. Here we describe another data representation, compressed miniscope (CMS), which allows the support of operations such as "for symmetry reasons it is sufficient to consider only ..." and "since A is a special case/alphabetic variant of B we conclude that ...."

We support these operations on PC formulas, but we first illustrate what is at stake with the propositional calculus. Let us start with a propositional calculus formula P_0 (and to motivate the operations we assume that it is a conjecture). Using a well-known formula, we can transform P_0 into an equivalent formula P_1

such that P_1 is in CNF, say

$$P_1 = \bigwedge R_i, \qquad \text{with} \quad R_i = \bigvee(R_{ij}).$$

(To simplify the notation, we frequently use $\bigwedge R_i$ as an abbreviation for $\bigwedge(R_1, \ldots, R_n)$: $\bigwedge$ stands for "and"; $\bigvee(R_{ij})$ abbreviates $\bigvee(R_{i1}, \ldots, R_{im})$.)

P_1 may be simplified with the following rules:

(1) If $R_{ix} = R_{iy}$ (with $x \neq y$), then drop R_{iy} from R_i (whenever only one element of the disjunction remains, drop the or connective).
(2) If $R_{ix} = \sim R_{iy}$, then drop R_i from P_1.
(3) If each element R_{ix} in R_i has a corresponding element R_{jy} in R_j with $R_{ix} = R_{jy}$, then drop R_j from P_1.
(4) If $\sim R_i = R_j$, then $P_2 :=$ FALSE.
(5) If no R_i remains, then $P_2 :=$ TRUE.
(6) If only one R_i remains, then drop the $\bigwedge$ connective.

Whenever the resulting formula P_2 is a conjunction while P_0 is not, then we have decomposed the problem P_0 into subproblems that are, in general, easier to solve than P_0. Rule 1 simplifies subproblems. Rule 2 removes tautologies. Rule 3 removes a copy of a subproblem. (The generalization of Rule 3 to the PC will remove, for instance, subproblems that are introduced by inherent symmetries.) Rule 4 is a watchdog against nonsensical problems. Rule 5 makes these rules a special-case theorem prover.

The generalization of this transformer to PC input is the topic of the next section. (The translator INSURER applies the rules while transforming to CMS, the analog of CNF, instead of applying those rules afterward.) Rules 1–4 presuppose a test to check whether propositional constants are related in such a way that a rule may "fire." For example, Rule 1 requires only a test for the identity of R_{ix} and R_{iy}. The generalization of these tests to the PC is the topic of this section and amounts to the INSTANCE algorithm. INSTANCE thus plays the same role as a subsumption algorithm. However, whereas a subsumption algorithm works only with clauses as input, INSTANCE works on PC formulas.

A CMS formula is a closed miniscope PC formula (see [14]), with the additional properties that

(a) no two arguments of an "and" or "or" subformula are in an INSTANCE relationship,
(b) no two arguments of such a subformula are in a half-negated INSTANCE relationship (the two formulas F and G are in a half-negated INSTANCE relation when the negation of F, with "not" moved inward, is in an INSTANCE relationship with G), and
(c) each quantifier has a unique variable.

Apart from the necessity of being more precise about miniscope, we have to face the complication that the INSTANCE relationship is mentioned in the definition of CMS, while the INSTANCE algorithm definition presupposes that its two arguments will come from the CMS domain. Recursion will be the way of solving this dilemma.

We remind the reader that a formula is in miniscope when

(a) the only logical symbols are $\sim$, $\bigwedge$, and $\bigvee$, and $\sim$ can occur only in front of atomic formulas;

(b) an argument of a conjunction (disjunction) is not a conjunction (disjunction);
(c) the body of a universal (existential) quantified formula is not a conjunction (disjunction);
(d) if the body of a quantified formula is a conjunction (disjunction), then each argument of the conjunction (disjunction) should contain the quantifier variable;
(e) no permutation of a sequence of universal (existential) quantifiers invalidates the former rule.

We still have to refine this miniscope format to CMS. Let the AND/OR level of a formula be the maximum number of AND/OR connectives one may encounter by going from the top level to a literal terminal. Suppose CMS has been defined up to level $n - 1$ and INSTANCE has already been defined on this CMS subset. An n-AND/OR level miniscope formula is in CMS iff no arguments of its nth-level AND/OR connective are in the INSTANCE or half-negated INSTANCE relationship (which is defined, since those are, at most, of level $n - 1$).

Before continuing with the definition of INSTANCE, we present examples to clarify the direction in which we are going.

Example 1. $(x)\{A(a) \wedge A(x)\}$ is not miniscope.

Example 2. $A(a) \wedge (x)A(x)$ is not CMS because $A(a)$ is in the INSTANCE relationship with $(x)A(x)$.

Example 3. $\sim A(a) \wedge (x)A(x)$ is not CMS because $\sim A(a)$ is in the half-negated INSTANCE relationship with $(x)A(x)$.

Example 4. $(x)A(x)$ is in CMS.

Remark. Example 2 may be rewritten to $(x)A(x)$ and Example 3 rewrites to FALSE.

We proceed with the specification of INSTANCE. Its input consists of a pair (T, K) of CMS formulas, where we may think of T as a conjecture to be tested for being a special case and/or alphabetic variant of K.

The output of INSTANCE is Y or nil depending on whether or not T is an "instance" of K. An "instance" is, in fact, defined in the following in a procedural way by an algorithm whose main characteristic is that it is stronger than "implication." After its specification we show that INSTANCE(T, K) implies $\vdash K \rightarrow T$.

The first action of INSTANCE is to Skolemize K and to "anti-Skolemize" T; thus, existential quantifiers in K and universal quantifiers in T are removed by replacing the associated variables with fresh functions, while universal quantifiers in K and existential quantifiers in T remain. Those functions have variables that depend on the preceding universal and existential quantifiers, respectively.

Example. Consider the formula Z: $(x)(E\ y)P(x, y)$. When Z is the second argument K of INSTANCE, the Skolemization of Z yields $(x)P(x, F(x))$, with F as a fresh unary Skolem function. When Z is the first argument T of INSTANCE, then anti-Skolemization of Z yields $(E\ y)P(f, y)$ where f is a Skolem constant since there is no existential quantifier preceding (x).

LEMMA 1. *If $S_1(K)$ is the result of the Skolemization of K, and $S_2(T)$ the result of the anti-Skolemization of T, then $\vdash S_1(K) \rightarrow S_2(T)$ implies $\vdash K \rightarrow T$.*

PROOF. What is given, $\vdash S_1(K) \rightarrow S_2(T)$, is equivalent to

(1) $\vdash S_1(K) \rightarrow \sim S_1(\sim T)$,

because of the nature of Skolemization and anti-Skolemization. Obviously, (1) is equivalent to

(2) $\vdash \sim S_1(K) \vee \sim S_1(\sim T)$,

which again is equivalent to

(3) $\vdash \sim \{S_1(K) \wedge S_1(\sim T)\}$.

Since S_1 distributes over conjunctions, we see that (3) is equivalent to

(4) $\vdash \sim S_1(K \wedge \sim T)$.

The soundness theorem allows us to infer that (4) is equivalent to

(5) $S_1(K \wedge \sim T)$ is unsatisfiable.

When SN(F) denotes a Skolem-normal form of a first-order formula F, then (5) is equivalent to

(6) SN($K \wedge \sim T$) is unsatisfiable.

For example, when F is the formula $(x)(E\ y)(z)G(x, y, z)$, then $S_1(F) = (x)(z)G(x, s(x), z)$, with s being a fresh Skolem function, and SN(F) $= (E\ S)(x)(z)G(x, S(x), z)$, with S as a second-order-function variable. A model that would satisfy $S_1(F)$ would provide an interpretation for the function s that could be used for the interpretation of the function S. A similar argument applies for a model that would satisfy SN(F) because the function s is not constrained by the surrounding formula.

Applying, for instance, lemma 42a of [4], we obtain that (6) implies

(7) $K \wedge \sim T$ is unsatisfiable.

Remark. Invoking the axiom of choice gives us: (7) implies (6). However we do not need the opposite implication.

Applying the completeness theorem finally gives us

(8) $\vdash K \rightarrow T$. □

The next action of INSTANCE, after anti-Skolemizing and Skolemizing its two arguments, is to call a recursive support function, INS2, with

$$\text{INS2}(S_2(T), S_1(K), \text{nil}),$$

where the third argument stands for the set of free variables in $S(T)$ and $S(K)$. Since we start off with closed formulas, this set will be empty at the top-level invocation of INS2. The function INS2 will descend into subformulas of its first two arguments and, when quantifiers are encountered, will increment the variable set for recursive calls. When INS2 encounters atomic formulas, it will apply the unification algorithm after taking accumulated variables into account.

The output of INS2 is either

—NO, signifying that $S_2(T)$ is not an instance with respect to the INS2 procedure of $S_1(K)$ and will cause INSTANCE to return with nil, or
—a nonempty set of substitutions $\{\sigma\}$, where each substitution σ allows the inference $\vdash S_1(K) \rightarrow S_2(T)$ and will cause INSTANCE to return with Y.

At the top-level call of INS2, it is sufficient that INS2 returns with only one substitution to have INSTANCE reporting success. The following example shows that the power of INSTANCE increases when INS2 returns possible multiple substitutions.

We would certainly like it to be recognized that

$$S_2(T) = (E\ z)(A(p, z) \wedge B(z))$$

is an instance of

$$S_1(K) = (x)A(x, f(x)) \wedge (y)A(y, g(y)) \wedge B(g(p)).$$

The function INS2 will produce two (dependent) subtasks. The first one is a recursive call with arguments $A(p, z)$ and $S_1(K)$ (with variable z). The second call is on $[B(z)]$ and $S_1(K)$, where $[B(z)]$ indicates the constraining substitution involving z, generated by the first call. The first subtasks can produce two satisfying substitutions: $\{(x \leftarrow p, z \leftarrow f(p)), (y \leftarrow p, z \leftarrow g(p))\}$. If INS2 were to produce only the first $\langle x, z \rangle$ substitution, then the other subtask concerning $[B(z)]$ would fail.

We proceed by defining the support function INS2. The arguments ST, SK, and VAR initially receive the values $S_2(T)$, $S_1(K)$, and nil, respectively. We need the following notation:

(a) $X.\tau$ stands for performing the substitution τ on X. {A substitution is of the form $((x_1 \leftarrow s_1), \ldots, (x_n \leftarrow s_n))$ with x_i not in s_j and all x_i different; nil is the empty substitution}.
(b) $\tau + \sigma$ stands for the concatenation of the substitutions τ and $\sigma.\tau$ (a variable may not occur in σ as well as in τ at the left-hand side, and no left-hand-side variable occurs in any right-hand side).
(c) $P{*}\tau$ is the formula obtained by removing each quantifier in P for which the associated variable has a replacement prescription in τ; and subsequently, performing $P'.\tau$ on the formula P' obtained in the previous step.

A unifier has to be understood as a most general unifier in the sense of [12]. We have generalized the unification algorithm slightly to allow also the matching of formulas. For example, $\bigwedge (A(x), B(b, x))$ and $\bigwedge (A(a), B(y, z))$ will have the unifier $(x \leftarrow a, y \leftarrow b, z \leftarrow a)$.

```
INS2(ST, SK, VAR) :=
if ST and SK are unifiable, with respect to the free variables of ST and SK as given by VAR,
  with unifier σ
  then {σ}
  else
if SK = (x)Form(x)
  then INS2(ST, Form(x) VAR ∪ {x})
  else
if ST = (E x)Form(x)
  then INS2(Form(x), SK, VAR ∪ {x})
  else
if SK = ∨ (K_1, ..., K_n)
  then [Z := INSORK(nil, ST, (K_1, ..., K_n));
        if Z = nil then NO else Z]
  {Where INSORK is a recursive function defined as
   INSORK(σ, STST, (KK_j, ..., KK_n), VAR) :=
```

```
      if j = n + 1, thus all K_i have been treated already
      then σ
      else [τ := INS2(STST, KK_j, VAR);
            if τ = NO then nil
            else ∪ Zν
              where Z_ν = INSORK(ν + σ, STST*ν, (KK_{j+1}. ν, . . . , KK_n.ν))
              for ν in τ]}
   else
if ST = ∧ (T_1, . . . , T_n)
   then [Z := INSANDT(nil, (T_1, . . . , T_n), SK);
         {The function INSANDT works like INSORK in the former case.}
         if Z = nil then NO else Z]
   else
if SK = ∧ (K_1, . . . , K_n)
   then [W := nil;
         for each K_i do
           {Z := INS2(ST, K_i, VAR);
           if Z unequal NO then W := W ∪ Z};
         if W = nil then NO else W]
   else
if ST = ∨(T_1, . . . , T_n)
   then [W := nil;
         for each T_i do
           {Z := INS2(T_i, SK, VAR);
           if Z unequal NO then W := W ∪ Z};
         if W = nil then NO else W]
   else NO.
```

To prove the soundness of INSTANCE, we first have to prove an already announced property of INS2.

LEMMA 2. *If σ is a substitution in INS2(ST, SK, VAR), then* $\vdash SK * \sigma \rightarrow ST * \sigma$.

PROOF. We do case reasoning and induction on the length of the formulas.

Case 1. If SK and ST are unifiable with substitution σ then $\text{SK} * \sigma = \text{ST} * \sigma$ and, thus, obviously $\vdash \text{SK} * \sigma \rightarrow \text{ST} * \sigma$.

Case 2. Let σ be a substitution produced by INS2(ST, Form(x), VAR $\cup$ $\{x\}$). Thus we have

(1) $\vdash \text{Form}(x) * \sigma \rightarrow ST * \sigma$.

If x occurs at the left side of a component in σ, then $\text{Form}(x) * \sigma = (x)\text{Form}(x) * \sigma$ and we are done. If x does not occur then we apply generalization on (1).

Case 3. This case with ST = $(E\,x)$Form(x) runs parallel to the former case.

Case 4. Assume that σ is in INSORK(nil, ST, $(K_1, \ldots, K_n)$). We have to show $\vdash \bigvee (K_1, \ldots, K_n) * \sigma \rightarrow \text{ST} * \sigma$.

We proceed by induction on the INSORK calls. Thus, assume that $K_1, \ldots, K_{j-1}$ have been dealt with already and that σ has been obtained thus far with STST = ST $* \sigma$, $KK_i = K_i * \sigma$ and

(1) $\vdash \bigvee (K_1, \ldots, K_{j-1}) * \sigma \rightarrow \text{ST} * \sigma$.

Let τ be in INS2(STST, KK, VAR); thus we have

(2) $\vdash KK_j * \tau \rightarrow \text{STST} * \tau$,

which corresponds with

(3) $\vdash K_j * \sigma * \tau \to \mathrm{ST} * \sigma * \tau$.

Since we infer

(4) $\vdash \bigvee (K_1, \ldots, K_{j-1}) * \sigma * \tau \to \mathrm{ST} * \sigma * \tau$

from (1), we can combine (3) and (4) to finish the induction step. The base case is obvious since we have $\vdash$ FALSE $\to$ ST.

Case 5. This case with ST = $\bigwedge (T_1, \ldots, T_n)$ runs parallel again to the former case.

Case 6. Let σ be in INS2(ST, $\bigwedge (K_1, \ldots, K_n)$, VAR); thus there is a K_i with σ in INS2(ST, K_i, VAR). So we have

$$\vdash K_i * \sigma \to \mathrm{ST} * \sigma.$$

Since obviously

$$\vdash \bigwedge (K_1, \ldots, K_n) * \sigma \to K_i * \sigma,$$

we are done.

Case 7. This case with ST = $\bigvee (T_1, \ldots, T_n)$ runs parallel again to Case 6.

We have dealt with all cases of the INS2 function, thus we have confirmed the induction step. The base of the induction was dealt with in Case 1. □

LEMMA 3. *If σ is a substitution in INS2(ST, SK, VAR) then*

$$\vdash (Vk)SK \to (E\ Vt)ST,$$

where Vt and Vk are, respectively, the free variables of ST and SK that belong to VAR.

PROOF. According to Lemma 2 we have

(1) $\vdash \mathrm{SK} * \sigma \to \mathrm{ST} * \sigma$.

As a consequence of .

(2) $\vdash (Vk)\mathrm{SK} \to \mathrm{SK} * \sigma$,

(3) $\vdash \mathrm{ST} * \sigma \to (E\ Vt)\mathrm{ST}$,

(4) $\vdash \{((Vk)\mathrm{SK} \to \mathrm{SK} * \sigma) \to$
$[(\mathrm{ST} * \sigma \to (E\ Vt)\mathrm{ST}) \to$
$\{(\mathrm{SK} * \sigma \to \mathrm{ST} * \sigma) \to ((Vk)\mathrm{SK} \to (E\ Vt)\mathrm{ST})\}]\}$,

we conclude by deleting left-hand sides of implications in (4), using *modus ponens* with (1)–(3) as premises:

$$(Vk)\mathrm{ST} \to (E\ Vt)\mathrm{ST}.$$ □

Combining Lemmas 1–3, we obtain

THEOREM 1. *If T and K are closed, compressed miniscope, predicate calculus formulas and INSTANCE(T, K) holds, then $\vdash K \to T$; thus INSTANCE is sound.*

PROOF. According to Lemma 1, we have

$$\vdash S_1(K) \to S_2(T) \text{ implies } \vdash K \to T.$$

Since $(Vk)S_1(K) = S_1(K)$ and $(E\ Vt)S_2(T) = S_2(T)$, because $S_1(K)$ as well as $S_2(T)$ are closed formulas, application of Lemmas 2 and 3 gives the required result. □

Since implication is transitive,

$$\{(P \to Q) \land (Q \to R)\} \vdash (P \to R),$$

and INSTANCE is a stronger version of implication, one may wonder whether INSTANCE is transitive. Do INSTANCE(P, Q) and INSTANCE(Q, R) imply INSTANCE(P, R)?

THEOREM 2. *If P, Q, and R are in compressed miniscope, then INSTANCE (P, Q) and INSTANCE(Q, R) imply INSTANCE(P, R).*

PROOF. We introduce the following abbreviations: SP := $S_2(P)$, SR := $S_1(R)$, $Q_1 := S_1(Q)$, and $Q_2 := S_2(Q)$. There are substitutions μ_1 and μ_2 with μ_1 in INS2(SP, Q_1, nil) and μ_2 in INS2(Q_2, SR, nil). We have to show that there is a substitution μ in INS2(SP, SR, nil). We induct on the number of quantifiers and connectives in P, Q, and R.

The base case is trivial since μ_1 and μ_2 must be empty. Therefore, we have SP = $Q_1 = Q_2$ = SR, which ascertains that the empty substitution belongs to INS2(SP, SR, nil).

The induction step requires 12 different cases concerning an additional quantifier { ( ) or (E) } or connective {$\land$ or $\lor$} in P, Q, or R, respectively. We only consider an additional ( ) quantifier and an additional $\lor$ connective in Q, since the other cases are similar.

Case ( ) *in Q.* Let the variable bound by the additional quantifier be x. When x is not bound in μ_1, we let μ be the substitution obtained by induction, yielded by the triple P, QQ, R where $QQ := Q^*(x \leftarrow c)$ and c is the anti-Skolem constant generated by the production of Q_2. Otherwise, when x is bound in μ_1 to, say, c_2, we take for μ the substitution obtained by induction, yielded by the triple P, QQ, R where $QQ := Q^*(x \leftarrow c_2)$.

Case $\lor$ *in Q.* There is a Q in $\lor(Q_1, \ldots, Q_n)$ which produces the μ_2 substitution. There will certainly be a substitution μ_1^* produced by INS2(P, Q_i, nil). So, we take for μ the substitution inducted by the triple P, Q_i, R. □

Remark 1. The transitivity of INSTANCE has relevance in practice. Suppose we have the conjunction $A \land B \land C$. Assume we have INSTANCE(A, B). Thus, we can replace $A \land B \land C$ with the equivalent formula $B \land C$. Suppose we have also INSTANCE(C, A). Transitivity allows us to "forget" A, because we are certain to have INSTANCE(C, B), allowing us to reduce the conjunction to B.

Remark 2. The definition of CMS given in the beginning of this section mentions INSTANCE. As long as a similar INSTANCE procedure satisfies our theorems, we can accept another delineation of a subset of the PC claiming the name CMS. A uninteresting example would be INSTANCE(T, K) iff $T = K$. Since all results of the next section refer only to Theorem 1 of this section, they generalize immediately to other, stronger versions of INSTANCE.

4. *INSURER*

INSURER is the theorem prover preprocessor that expects for its input a problem, specified as a closed predicate calculus formula, and that tries to rewrite it in an equivalent formula with the leading connective "and" (while restraining itself from

rewriting, for instance, P into $P \wedge P$). If the output is a conjunction, then the subformulas of the conjunction can be seen as subproblems. When all of them can be solved, the original problem has been dealt with. The main issue of this section is whether INSURER gives a maximal decomposition into independent subproblems.

As we have already stated in [2], the set of rewrite rules that make up INSURER is, coincidentally, a subset of the rewrite rules that make up to PC–CNF translator. Despite their similarity, their behavior is substantially different. Whereas a PC–CNF translator produces CNF, the procedure INSURER produces equivalent, closed PC formulas. (In addition, note that a PC–CNF translator is usually applied to the negation of a conjecture, whereas INSURER is applied to the conjecture itself.) Since we shall argue that this PC–CNF translator remedies deficiencies of the one described in [3], [11], and [10], we first describe the PC–CNF translator and discuss its properties.

The translator consists of the following sequence:

(1) eliminate "if, . . . , then" and "if and only if";
(2) move "not" inward;
(3) push quantifiers to the right;
(4) eliminate existential quantifiers by Skolem functions and delete universal quantifiers; and
(5) distribute "and" over "or".

Steps 1, 2, and 4 are standard. Steps 3 and 5 invoke INSTANCE. We describe each step briefly.

(1) Let t_1 be the procedure that eliminates implications and equivalences:

t_1 (formula) :=
if formula = $A \rightarrow B$ **then** $\bigvee (\sim t_1(A), t_1(B))$ **else**
if formula = $A \leftrightarrow B$ **then** $\bigwedge (\bigvee(\sim t_1(A), t_1(B)), \bigvee(t_1(A), \sim t_1(B)))$ **else**
if formula is atomic **then** formula **else**
the result of descending recursively in formula.

Remark. In our implementation, we always introduce abbreviations for (sub)formulas. This allows us to associate properties with (sub)formulas and, in particular, it will prevent subsequent reprocessing of the duplicated arguments of an equivalence.

(2) Let t_2 be the procedure that moves the negation symbol inward:

t_2 (formula) :=
if formula = $\sim\sim A$ **then** $t_2(A)$ **else**
if formula = $\sim(x)A$ **then** $(E\,x)t_2(\sim A)$ **else**
if formula = $\sim(E\,x)A$ **then** $(x)t_2(\sim A)$ **else**
if formula = $\sim(\bigvee(A_1, \ldots, A_n))$ **then** $\bigwedge(t_2(\sim A_1), \ldots, t_2(\sim A_n))$ **else**
if formula = $\sim(\bigwedge(A_1, \ldots, A_n))$ **then** $\bigvee(t_2(\sim A_1), \ldots, t_2(\sim A_n))$ **else**
if formula is atomic or $\sim$(atomic) **then** formula **else**
the result of descending recursively in formula.

Remark. In our implementation of this step, we always associate each (sub)formula with its negation, while the negated formula is associated with the original formula. This only doubles the number of formulas we must deal with, but saves time in subsequent processing, as we point out in the following.

(3) Let t_3 be the procedure that moves quantifiers to the right:

```
t₃ (formula) :=
  {let XX, YY be metavariables standing for ∧ or ∨}
if formula = XX(A₁, . . . , Aₙ) then T3AO(XX(t₃(A₁), . . . , t₃(Aₙ))),
  where the function T3AO is explained below;
else
  {let (Q x) be (x) or (E x)}
if formula = (Q x)A then
  let B = t₃(A);
  if B = XX(B₁, . . . , Bₙ) then
    if [(Q x) = (x) and XX = ∧] or [(Q x) = (E x) and XX = ∧] then
      t₃(XX(C₁, . . . , C_K, (Q x)D₁, . . . , (Q x)D_l)),
      where Cᵢ and Dᵢ are the formulas Bᵢ that do not and do contain the variable x,
      respectively;
    else
    if there is a Bᵢ that does not contain x then
      t₃(XX(C₁, . . . , C_K, (Q x)XX(D₁, . . . , D_l)),
      where Cᵢ and Dᵢ are the formulas Bᵢ that do not and do contain the variable x,
      respectively;
    else
    if there is a Bᵢ with Bᵢ = YY(B_i1, . . . , B_im) and
      [(Q x) = (x) and YY = ∧] or [(Q x) = (E x) and YY = ∨] then
      t₃(YY((Q x)XX(B_i1, C₁, . . . , C_K), . . . , (Q x)XX(B_im, C₁, . . . , C_K))),
      where Cⱼ = Bⱼ unless j = i
    else (Q x)XX(B₁, . . . , Bₙ)
  else
  if B = (Q y)C then (Q y)t₃((Q x)C) else (Q x)B
else formula.
```

We have to explain the support function T3AO. This function deals with conjunctions/disjunctions where the subformulas have been treated already by t_3. T3AO attempts to remove subformulas from the conjunction/disjunction, and it also tries to collapse its argument completely. It invokes INSTANCE for these attempts. More precisely, it attempts to "flatten" its argument; to remove subformulas; and to collapse the remaining arguments.

To facilitate the description of these actions inside T3AO, we assume that the input argument is a conjunction. The flattening operation works as follows:

```
flatten(formula) :=
if formula = ⋀(A₁, . . . , TRUE, . . . , Aₙ) then
  flatten(⋀(A₁, . . . , Aₙ))
else
if formula = ⋀(A₁, . . . , FALSE, . . . , Aₙ) then
  FALSE
else
if formula = ⋀(A₁, . . . , ⋀(B₁, . . . , Bₘ), . . . , Aₙ) then
  flatten(⋀(A₁, . . . , B₁, . . . , Bₘ, . . . , Aₙ))
else formula.

remove-argument(⋀(A₁, . . . , Aₙ)) :=
if i ≠ j and INSTANCE(Aᵢ, Aⱼ) for some i and j then
  remove-argument(⋀(B₁, . . . , Bₘ)) where {Bk} = {Ak | k ≠ i}
else ⋀(A₁, . . . , Aₙ).

collapse-argument(⋀(A₁, . . . , Aₙ)) :=
if INSTANCE(t₂(~Aᵢ), Aⱼ) for some i and j then
  FALSE
else ⋀(A₁, . . . , Aₙ).
```

Remark. The tests applied in the subfunction collapse-argument benefit substantially from the associations between a formula and its negation that we introduced in phase two. In our implementation, we store the outcome of an INSTANCE test at its first argument to prevent redoing these calculations when the arguments are reencountered (say, because of a distribution of $\wedge$ over $\vee$).

(4) Let t_4 be the procedure that eliminates existential quantifiers; we execute t_4(output of step 3, nil), where the second argument, here nil, stands for the set of universally quantified free variables in the first argument.

t_4(formula, $\{x_1, \ldots, x_n\}$) :=
if formula = $(E\ x)A$ **then**
  $t_4(A(x \leftarrow f(x_1, \ldots, x_n)), \{x_1, \ldots, x_n\})$, where f is a fresh Skolem function and each occurrence of x in A is replaced by $f(x_1, \ldots, x_n)$,
else
if formula = $(x)A$ **then**
  $t_4(A, \{x, x_1, \ldots, x_n\})$
else the result of applying t_4 recursively on subformulas.

(5) Let t_5 be the function that distributes $\wedge$ over $\vee$

t_5(formula) :=
if formula = $\bigwedge(A_1, \ldots, A_n)$ **then**
  T5A($\bigwedge(t_5(A_1), \ldots, t_5(A_n))$)
else
if formula = $\bigvee(A_1, \ldots, A_n)$ **then**
  T5O($\bigvee(t_5(A_1, \ldots, A_n))$)
else formula.

The support function T5A works like T3AO; see above. The support function T5O first does the activity like T3AO and subsequently checks whether a subformula is a conjunction. If so, the function t_5 is applied again to the result of distributing $\wedge$ over $\vee$.

The main difference between this PC–CNF translator and those described in [3], [10], and [11] is the incorporation of INSTANCE. The translator in [3] is based on first producing prenex normal form, which is done with rules 1 and 2, followed by pushing quantifiers to the *left*. Consequently, Skolem functions may be introduced with an unnecessary number of arguments, for example, $(x)\{A(x) \vee (E\ y)B(y)\}$ will be transformed into $(x)(E\ y)\{A(x) \vee B(y)\}$ leading to the CNF $A(x) \vee B(f(x))$. Instead, it can be transformed into $(x)A(x) \vee (E\ y)B(y)$, leading to the simpler form: $A(x) \vee B(g)$. The translator in [10] lacks the distribution of $\wedge$ over $\vee$, or vice versa, in the scope of a quantifier and its preparatory "flatten" operation, while the translator in [11] lacks, in addition, the appropriate distribution of quantifiers over connectives. Consequently, these translators may be forced to generate Skolem functions with too many arguments, and/or to produce too many clauses and/or too many literals.

In [10, Theorem 1.5.1], it is shown that the translator preserves unsatisfiability. Combining this result with application of the soundness theorem in Section 3 on the rules containing INSTANCE-related rewriting leads also to preservation of unsatisfiability by the translator detailed here.

We define INSURER as the subset of the PC–CNF translator rules that is obtained by omitting rule 4 concerning the elimination of existential and universal quantifiers. INSURER produces closed PC formulas, and, since all transformations concern equivalences, we have

LEMMA 1. *If Q := INSURER(P) then $\vdash P \leftrightarrow Q$.*

While the input format of INSURER is the unrestricted PC, the next observation concerns its output format.

LEMMA 2. *INSURER maps PC formulas into compressed miniscope.*

PROOF. The miniscope property is assured since INSURER checks recursively that terms of disjunctions/conjunctions in the matrix of a quantified formula contain the variable of the quantifier.

CMS is partially defined in an operational way by reference to INSTANCE (which eliminates redundancies, as recognized by INSTANCE, in conjunctions and disjunctions). Since INSTANCE is incorporated at the proper places in INSURER, we are assured that the compressed feature is satisfied as well. □

INSURER turns out to be a special-case theorem prover. It can recognize at least ground tautologies.

LEMMA 3. *If Q is a valid PC formula without quantifiers, then INSURER(Q) = TRUE.*

PROOF. Since the output of INSURER is in miniscope, INSURER(Q) is TRUE, or FALSE, or of the form ⟨literal_atom⟩, $\bigvee$⟨literal_atom⟩*, or a conjunction with each conjunct being a ⟨literal_atom⟩ or $\bigvee$⟨literal_atom⟩*.

The case FALSE is prohibited by Lemma 1. The case ⟨literal_atom⟩ contradicts the validity assumption because the value "false" may be assigned to ⟨literal_atom⟩.

Suppose the output is of the form $\bigvee$⟨literal_atom⟩*, and thus like $\bigvee(O_1, \ldots, O_k)$. If all O_i are positive (or all are preceded by a negation sign), then we have a contradiction since we can assign all O_i the value "false" ("true"). Thus, we can rewrite the disjunction as follows: $\{\bigvee(P_1, \ldots, P_n)\} \vee \{\bigvee(\sim N_1, \ldots, \sim N_m)\}$. No P_i can be equal to a N_j, since that would have been recognized in step 5 of INSURER by INSTANCE. Again, we get a contradiction with validity by assigning all P_i the value false and all N_j the value "true". Consequently, $\bigvee$⟨literal_atom⟩* is ruled out.

Applying the former cases on every conjunct in a conjunction eliminates this case as well. Thus INSURER(Q) can only be TRUE. □

A generalization to the monadic predicate calculus turns out not to hold. A counterexample is

$$(x)Q(x) \leftrightarrow [(y)\{Q(y) \vee P(y)\} \wedge (z)\{Q(z) \vee \sim P(z)\}].$$

This is valid and will be translated into

$$(x)Q(x) \vee (y)\{\sim Q(y) \vee P(y)\} \vee (z)\{\sim Q(z) \vee \sim P(z)\},$$

instead of TRUE.

INSURER is a recognizer of independent subproblems, since INSURER is "strongly motivated" to rewrite its input into an equivalent (Lemma 1) formula which is a conjunction. In case the output is a conjunction, each of the two arguments is independent of each other with respect to the implicative testing of INSTANCE. We described INSURER as being strongly motivated since it is not possible to prove that INSURER gives a maximal conjunctive decomposition. Even an atomic formula P can be equivalently rewritten into the conjunction $(P \wedge Q) \vee (P \vee \sim Q)$ for an arbitrary Q. This conjunction, however, is an example

of case reasoning because it can be rewritten as

$$(Q \rightarrow P) \wedge (\sim Q \rightarrow P),$$

embodying the maxim: In order to prove P it is sufficient that Q as well as its negation implies P.

We are going to show that if a nonconjunctive output (or output component) Q of INSURER can be rewritten into an equivalent conjunction $\bigwedge R_i$, then this conjunction $\bigwedge R_i$ embodies case reasoning on Q; that is, $\bigwedge R_i$ is a disguised version of $\bigwedge(Q \vee R_i')$ with $\bigwedge R_i'$ equivalent to false. Since one cannot expect that INSURER takes the initiative to do case reasoning, we conclude that INSURER produces maximal decompositions. First, we deal with supporting lemmas. The next lemma deals with the quantifier-free case and assumes a particular format for the R_i's.

LEMMA 4. *Assume*

—*Q and R to be quantifier-free CMS formulas*;

—*Q is not a conjunction, $Q = \bigvee(Q_1, \ldots, Q_q)$ (possibly with $q = 1$), and is falsifiable*;

—*R is a conjunction and thus $R = \bigwedge R_i$, with $R_i = \bigvee(R_{i1}, \ldots, R_{ir})$ (possibly $r = 1$)*;

—$\vdash Q \leftrightarrow \bigwedge(Q \vee R_i)$.

Let R_i' be defined as

R_i' := **if** *$R_{ip} = \sim Q_s$ for some p and s* **then** *TRUE* **else** *$\bigvee(R_{ij}')$, with R_{ij}' :=* **if** *$R_{ij} = Q_s$ for some s* **then** *FALSE* **else** *R_{ij}.*

Then we have

$\vdash Q \leftrightarrow \bigwedge(Q \vee R_i')$ and
$\vdash \sim\bigwedge R_i'$.

PROOF. The construction of the R_i' allows us to conclude for each i that

$$\vdash Q \vee R_i \leftrightarrow Q \wedge R_i'.$$

Therefore, we immediately have $\vdash Q \leftrightarrow \bigwedge(Q \vee R_i')$.

Assume that $\sim\bigwedge R_i'$ is falsifiable. Thus $\bigwedge R_i'$ is satisfiable. Therefore, there is an assignment that makes VAL($\bigwedge R_i'$) = true. Since, owing to its construction, $\{R_i'\}$ does not share a literal with Q, we can extend this assignment without a constraint to cover Q as well. An extension that makes VAL(Q) = false produces a contradiction with $\vdash Q \leftrightarrow \bigwedge(Q \vee R_i')$. □

The next lemma deals with the occurrence(s) of quantifiers. It shows that a conjunction $\bigwedge R_i$, equivalent to Q, is a "mystified" version of a conjunction $\bigwedge(Q \vee R_i')$, which holds that $\bigwedge R_i'$ is equivalent to false. We reduce the situation to the quantifier-free case—ultimately allowing the application of Lemma 4—by stripping away a quantifier at the time. We introduce construction rules that, using a pair $\langle Q, \bigwedge R_i \rangle_n$, produce another related pair, $\langle Q', R_i' \rangle_{n+1}$, which contains one quantifier less.

LEMMA 5. *If*

(1) *Q is a falsifiable formula in CMS and not a conjunction,*

(2) *R is in CMS and a conjunction and thus $R = \bigwedge R_i$ with each R_i not a conjunction, and*

(3) $\vdash Q \leftrightarrow \bigwedge R_i$,

then $\bigwedge R_i$ is directly or indirectly an example of case reasoning on Q; that is, $\bigwedge R_i$ is obtained from Q by (possibly repeatedly) replacing q's in Q by formulas of the form $\{(q \vee p) \wedge (q \vee \sim p)\}$.

PROOF. FROM $\vdash Q \leftrightarrow \bigwedge R_i$ it is easy to see that

(1) $\vdash Q \leftrightarrow \bigwedge(Q \vee R_i)$.

We have

$Q = Q_1 \vee \cdots \wedge Q_q$ (possibly $q = 1$),
$R_i = R_{i1} \vee \cdots \vee R_{ir}$ (possibly $r = 1$).

We define R_{ix0} with

R_{ix0} := **if** there is a Q_y with INSTANCE(Q_y, R_{ix}) and INSTANCE(R_{ix}, Q_y), **then** FALSE **else** R_{ix}.

This allows us to define R_{i0} as $R_{i0} := \bigvee(R_{ix0})$ while deleting the R_{ix0}'s equal FALSE. We still have

(2) $\vdash Q \leftrightarrow \bigwedge(Q \vee R_{i0})$.

We have two possibilities:

(i) $\vdash \sim\bigwedge R_{i0}$, thus, we are dealing with an immediate example of case reasoning,

or

(ii) not $\vdash \sim\bigwedge R_{i0}$.

We proceed with case (ii). From (2) and by defining $Q_0 := Q$, we get

(3) $\vdash Q_0 \leftrightarrow \bigwedge(Q_0 \vee R_{i0})$,

and from (3)

(4) $\vdash \bigwedge R_{i0} \rightarrow Q_0$.

By Lemma 4, we can conclude that (3) and (4) should contain at least one quantifier.

We construct a terminating sequence of $\{Q_n\}$ and $\{R_{in}\}$ fulfilling (1), ultimately leading to case (i), by stripping away quantifiers. Assume that $\{Q_n\}$ and $\{R_{in}\}$ have already been constructed.

Construction Rule 1. If there is an R_{jn} such that $R_{jn} = (x)R_{j1n} \vee R_{j2n} \vee \cdots \vee R_{jrn}$, and we have a derivation of Q_n in which the full power of the universal quantifier is not used (i.e., the derivation tree for Q can be modified such that all occurrences of R_{jn} on leaf positions can be replaced by instantiations for x in R_{jn}), we define

$$\begin{aligned} Q_{n+1} &:= Q_n, \\ R_{in+1} &:= R_{in} \quad \text{for} \quad i \neq j, \\ R_{jn+1} &:= \bigwedge\{R_{j1n}(u_k) \vee R_{j2n} \vee \cdots \vee R_{jrn}\} \end{aligned}$$

where $\{u_k\}$ is the finite set of required instances for $(x)R_{j1n}(x)$ to derive Q_n. So we have

$$\begin{array}{ccc} \bigwedge R_{in} & \longrightarrow & Q_n \\ \downarrow & & \updownarrow \\ \bigwedge R_{in+1} & \text{———} & Q_{n+1} \end{array}$$

It is easy to see that we still have

$$\vdash Q_{n+1} \leftrightarrow \bigwedge(Q_{n+1} \vee R_{in+1}).$$

An example of the applicability of Construction Rule 1 is

$$Q(a, b) \leftrightarrow [\{Q(a, b) \vee Q(a, a)\} \wedge \{Q(a, b) \vee (x)(\sim Q(x, x) \vee Q(x, b))\}].$$
$$R_{10} = Q(a, a);\ R_{20} = (x)(\sim Q(x, x) \vee Q(x, b));$$

the finite set of necessary instances is here $\{a\}$. Thus in the next round we have

$$Q(a, b) \leftrightarrow [\{Q(a, b) \vee Q(a, a)\} \wedge \{Q(a, b) \vee \sim Q(a, a) \vee Q(a, b)\}],$$

which leads to the base case.

Construction Rule 2. If Q_n is of the form $(x)Q_{1n} \vee Q_{2n} \vee \cdots \vee Q_{qn}$, then weaken Q by instantiating x with a new constant c and thus define

$$Q_{n+1} := Q_n(c) \vee Q_{2n} \vee \cdots \vee Q_{qn},$$

and

$$R_{in+1} := R_{in}.$$

So we have

$$\begin{array}{ccc} \bigwedge R_{in} & \longrightarrow & Q_n \\ \updownarrow & & \downarrow \\ R_{in+1} & \longrightarrow & Q_{n+1} \end{array}$$

By induction on i, it can be shown that we still have

$$\vdash Q_{n+1} \leftrightarrow \bigwedge(Q_{n+1} \vee R_{in+1}).$$

An example of the applicability of Construction Rule 2 is

$$(x)Q(x, a) \leftrightarrow [\{(x)Q(x, a) \vee (y)(z)[P(y, z) \vee Q(y, z)]\} \wedge$$
$$\{(x)Q(x, a) \vee (y)(z)[\sim P(y, z) \vee Q(y, z)]\}],$$

replacing $(x)Q(x, a)$ by $Q(c, a)$. Thus in the next round we have

$$Q(c, a) \leftrightarrow [\{Q(c, a) \vee (y)(z)[P(y, z) \vee Q(y, z)]\} \wedge$$
$$\{Q(c, a) \vee (y)(z)[\sim P(y, z) \vee Q(y, z)]\}].$$

Construction Rule 3. If Q_n is of the form $(E\ x)Q_{1n}(x) \vee Q_{2n} \vee \cdots \vee Q_{qn}$ and we have a derivation of (4) that allows us to strengthen Q_n by instantiating x with a constant c, then define

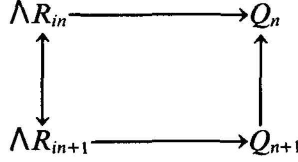
$$R_{in+1} := R_{in}$$

and

$$Q_{n+1} := Q_{1n}(c) \vee Q_{2n} \vee \cdots \vee Q_{qn}.$$

Q_{n+1} may not be in CMS any more, since $Q_{1n}(c)$ can be a conjunction. If so, we obtain as many examples of (1) as there are terms in the conjunction by distributing

"and" over "or." So we have



Whether we have to distribute $\wedge$ over $\vee$ or not, we still have (for each conjunct)

$$\vdash Q_{n+1} \leftrightarrow \wedge(Q_{n+1} \vee R_{in+1}).$$

An example of the applicability of Construction Rule 3 is

$$(E\ x)Q(a, x) \leftrightarrow [\{(E\ x)Q(a, x) \vee (y)(P(y) \vee Q(a, y))\} \wedge \{(E\ x)Q(a, x) \vee (E\ z){\sim}P(z)\}].$$

We obtain

$$Q_0 = (E\ x)Q(a, x),$$
$$R_{10} = (y)\{P(y) \vee Q(a, y)\},$$

and

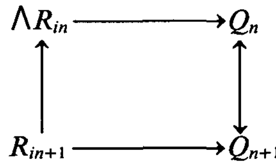
$$R_{20} = (E\ z){\sim}P(z).$$

By application of Construction Rule 4 (below), we get, for instance, $Q_1 = Q_0$, $R_{11} = R_{10}$ and $R_{21} = {\sim}P(c)$. Since $\wedge(R_{11}, R_{21})$ not only allows us to derive Q_1 but also $Q(a, c)$, application of Construction Rule 3 leads to $Q_2 = Q(a, c)$.

Construction Rule 4. If there is a R_{jn} such that $R_{jn} = (E\ x)R_{j1n}(x) \vee R_{j2n} \vee \cdots \vee R_{rjn}$, then strengthen R_{jn} by instantiating x with a new constant c and thus define

$$Q_{n+1} := Q_n,$$
$$R_{in+1} := R_{in} \quad \text{for} \quad i \neq j,$$
$$R_{jn+1} := R_{j1n}(c) \vee R_{j2n} \vee \cdots \vee R_{jrn}.$$

As with Construction Rule 3, R_{jn+1} may not be in CMS when $R_{j1n}(c)$ is a conjunction. By distributing "and" over "or" we get more $\{R_{in+1}\}$'s than $\{R_{in}\}$'s. So we have



Also in this last case, we still have (for each conjunct)

$$\vdash Q_{n+1} \leftrightarrow \wedge(Q_{n+1} \vee R_{in+1}).$$

For an example of applicability of Construction Rule 4, see the example in Construction Rule 3 above.

When a construction rule is applicable, we recheck whether the condition of case (i) holds, which leads to an example of case reasoning.

It remains to show that at least one construction rule applies. We immediately can rule out the cases in which Q_n contain a universal quantifier or $\wedge R_{in}$ an existential quantifier, since applicability of Construction Rule 2 or 4, respectively, depends only on these syntactic features. Thus, we are left with Q_n containing an

existential quantifier and/or $\wedge R_{in}$ containing a universal quantifier. Assume that Q contains an existential quantifier. Thus we have

(5) $\vdash \wedge R_{in} \rightarrow \{(E\ x)Q_{1n}(x) \vee Q_{2n} \vee \cdots \vee Q_{qn}\}$.

A derivation tree for (5) can be modified into a derivation tree for

$$\wedge R_{in} \rightarrow \{Q_{1n}(c) \vee Q_{2n} \vee \cdots \vee Q_{qn}\},$$

where c is fresh constant, by propagating modifications from the root to the leaves, unless a leaf of the tree is of the form $(E\ y)R_{jn}(y)$. This, however, we have already ruled out since it would lead to applicability of Rule 4.

The case in which $\wedge R_{in}$ contains a universal quantifier leads to a similar contradiction.

This process halts because we start with a finite number of quantifiers, and we strip off a quantifier each time a construction rule applies.

Thus we have shown that the original situation was a "disguised" form of case reasoning. □

Finally, we rephrase Lemma 5 as

THEOREM. *Let X be the output of INSURER when the output is not a conjunction or else an arbitrary member of the conjunction. INSURER produces a maximal conjunctive decomposition in the sense that there is no equivalent conjunctive decomposition of X, when we disregard case reasoning on X (see Lemma 5 above for the notion of case reasoning).*

PROOF. Apply Lemma 5. □

5. *Interplay between INSURER and INSTANCE*

INSURER, INSTANCE, a connection graph resolution-based contradiction recognizer, a PC–CNF translator, and a definition opener were imbedded in a "fixed" regime. Input for the prover consists of axioms, supporting theorems (proof sequence is not taken into account), definitions (again, without sequence), and the conjecture. For the next description, we should remember that activation of the connection graph component should be postponed at all costs.

Roughly, this supervisor triggers the following activities:

Step 1. **If** the conjecture in an INSTANCE of an axiom, a theorem, or an already proven theorem (see Step 2)
then return with success.

Step 2. **If** the conjecture, using INSURER, decomposes into the subproblems $C_1, \ldots, C_n$
then for each C_i go (recursively) to Step 1
  if the value returned for treating C_i is successful
  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 (nonrecursive) **then** replace in the conjecture each literal in which the predicate occurs by 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, invoke the resolution-based contradiction recognizer, and return its value (a resource parameter ensures termination).

(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) $(H)(\mathrm{SUBGR}(H) \leftrightarrow [\wedge\ (E\,x)H(x)\ (x)(y)\{H(x) \wedge H(y) \rightarrow H(xy)\}\ (x)\{H(x) \rightarrow H(I(x))\}])$
(7) $(H1)(H2)\,(\mathrm{SETEQ}(H1, H2) \leftrightarrow (x)(H1(x) \leftrightarrow H2(x)))$
(8) $(g)(X)(H)\,(\mathrm{COSET}(g, X, H) \leftrightarrow [\wedge\ \mathrm{SUBGR}(H)\ (x)\{X(x) \leftrightarrow (E\,y)(H(y) \wedge x = yg)\}])$
(9) $(g)(X)(H)\,(\mathrm{COSET}(g, X, H) \rightarrow [H(g) \leftrightarrow \mathrm{SETEQ}(X, H)])$

FIG. 1. Axioms (1)–(5) (not minimal), definitions (6)–(8), and a theorem (9) from group theory.

This is a simple-minded supervisor and made only to demonstrate the effectiveness of INSTANCE and INSURER. We have used a more sophisticated supervisor, which can cope with recursive definitions, in the context of program verification. However, much remains to be desired. An attractive alternative would be to implement the supervisor as a multiprocess scheduler. Then, the overall structure of the cooperating specialists would be more transparent. It would facilitate the addition of new specialists, and open the way to pseudoparallel processing, which, but for the lack of available languages like QLISP, INTERLISP, and MAGMA-LISP, would be possible.

6. *Implementation Results*

The first example comes from group theory; see Figure 1 for axioms (1)–(5), definitions (6)–(8), and theorem (9). The axioms (1)–(5) do not constitute a minimal characterization of a group. A subset of a group is represented by a predicate variable. The predicate SUBGR, which expresses the property of a subgroup, is therefore of second order. Equality of subsets is expressed by SETEQ in (7). The notion of a right coset is defined by (8); COSET(g, X, H) should be read as "X is the right-coset with respect to the subgroup H and the group element g." The predicates SETEQ and COSET are also of second order. Theorem (9) expresses that the element g belongs to the subgroup H iff H is equal to the g-H-coset.

Direct translation of (1)–(8) and the negation of (9) into conjunctive normal form yields 39 clauses with all together 109 intervals. INSURER, however, recognizes that (9) can be decomposed into

$$(10)\quad (g)(H)(H(g) \vee (X)]\{\vee\ {\sim}\mathrm{COSET}(g, X, H)\ {\sim}\mathrm{SETEQ}(X, H)\})$$

and

$$(11)\quad (g)(H)({\sim}H(g) \vee (X)\{\vee\ {\sim}\mathrm{COSET}(g, X, H)\ \mathrm{SETEQ}(X, H)\}).$$

Working on (10), the definitions of COSET, SETEQ, and SUBGR are, respectively, substituted. The result is negated and, together with (1)–(5), translated into conjunctive normal form yielding 14 clases with 23 literals. After removing COSET and SETEQ in (11), it turns out that INSURER applies again, splitting up (11) into two subproblems. Each one ends up with 13 clauses and 20 literals. Although

(1) $(x)(y)\,(\mathrm{Sd}(y) \rightarrow \mathrm{Sd}(\mathrm{merge}(x, y)))$
(2) $(x)(y)(u)\,\{(\mathrm{Sd}(y) \wedge \mathrm{Same}(x, y)) \rightarrow \mathrm{Same}(\mathrm{cons}(u, x), \mathrm{merge}(u, y))\}$
(3) $(x)\,(\mathrm{Equal}(x, \mathrm{nil}) \rightarrow R(x, \mathrm{nil}))$
(4) $(x)\,(\sim\mathrm{Equal}(x, \mathrm{nil}) \rightarrow \mathrm{Equal}(x, \mathrm{cons}(\mathrm{car}(x), \mathrm{cdr}(x))))$
(5) $(x)(u)(v)\,((\mathrm{Equal}(x, u) \wedge \mathrm{Same}(u, v)) \rightarrow \mathrm{Same}(x, v))$
(6) $(x)(y)\,(\mathrm{R}(x, y) \leftrightarrow (\mathrm{Same}(x, y) \wedge \mathrm{Sd}(y)))$
(7) $(x)(E\,y)\,(\bigwedge(\mathrm{Equal}(x, \mathrm{nil}) \rightarrow \mathrm{R}(x, y))\ ((\sim\mathrm{Equal}(x, \mathrm{nil}) \wedge \mathrm{R}(\mathrm{cdr}(x), \mathrm{sort}(\mathrm{cdr}(x)))) \rightarrow \mathrm{R}(x, y)))$

FIG. 2. Formulas used by Green to generate a sorting algorithm (with an answer predicate; see [7]). The formulas are axioms (1)–(5), definition (6), and conjecture (7), respectively.

our resolution component is not able to handle these three subproblems, the chance of finding a solution has increased "infinitely" when compared with the nondecomposed situation.

INSURER can also handle the sorted predicate calculus that was described in [1]. The same coset example formulated in sorted predicate calculus, without decomposition, yields 28 clauses with 61 literals. INSURER also finds here three subproblems, each having 12 clauses with 16, 14, and 14 literals, respectively, creating a significant reduction again. However, the connection graph resolution component, in the meantime extended with paramodulation facilities, still cannot handle them. (Instead of relying on paramodulation, we consider adding an equality specialist to the deductive community.)

The next example was taken from [7] and was already worked on as reported in [1] (see Figure 2). It was originally used in [7] for illustrating automatic programming. A sorting algorithm was generated by adding an "answer-predicate" to the negated conjecture and submitting all the formulas to the QA3 resolution theorem prover. Green [7] admits that the axioms are "tuned" for the algorithm generation. The conjecture contains, for instance, the function "sort," which is not referred to by the other axioms. In fact, from the axioms one can prove expression (7) by replacing "$R(\mathrm{cdr}(x), \mathrm{sort}(\mathrm{cdr}\,(x)))$" with "$(E\,z)R(\mathrm{cdr}(x), z)$", from which (7) can be inferred.

The main predicate is Sd, which expresses that its argument, a list, is sorted. The expression $R(x, y)$ signifies that the list y is a sorted permutation of the list x; Equal(x, y) signifies that the list x is identical with the list y; the empty list is indicated by nil. The function "merge" stands for merging a list with a sorted list such that a sorted list is the result. The function "cons" corresponds to adding an element to the front of a list. The functions "car" and "cdr," respectively, produce the first element and the remainder of a list.

INSURER will decompose the conjecture into two subproblems. When INSTANCE is not incorporated, eight subproblems will be found, six of which will be redundant. Subsequently, INSTANCE recognizes that one of the subproblems is an instance of axiom (3). The remaining subproblem is solved with definition substitution, as well as without definition substitution (by adding the definition to the axioms). In both cases a contradiction is found more easily than in the nondecomposed case (see Table I).

Table I shows the effectiveness of INSURER and INSTANCE. The numbers between brackets refer to values obtained when the sorted predicate calculus is used [1]. The g penetrance is defined as #(clauses in proof)/#(input + generated clauses). The QA3 values were taken from [7].

A later version of our theorem-proving complex incorporates the evaluation of a conjecture and/or subgoals in models (in order to prune the search tree).

TABLE I

Program and strategy	Input + generated clauses	*g* penetrance
QA3	286	0.091
Resolution only	38 (25)	0.579 (0.680)
+ INSURER and INSTANCE	28 (17)	0.785 (0.882)
+ Definition substitution	20 (12)	0.800 (0.917)

Predicates defined in a theory are proceduralized automatically after the application of INSURER to the body of the definition. For example, the notion of transitivity was encountered in the usual form:

$$(R)\{\mathrm{TRAN}(R) \leftrightarrow (x)(y)(z)[\{R(x, y) \wedge R(y, z)\} \rightarrow R(x, z)]\}.$$

INSURER rewrites the body into

$$(x)(y)[\sim R(x, y) \vee (z)\{\sim R(y, z) \vee R(x, z)\}].$$

This is an improvement, indeed, since the procedure that results from replacing the quantifiers by loops now has a core loop over z that is 33 percent cheaper.

Our final example consists of only one formula:

$$[\{(E\ x_1)(y_1)P(x_1) \leftrightarrow P(y_1)\} \leftrightarrow \{(E\ x_2)Q(x_2) \leftrightarrow (y_2)P(y_2)\}] \leftrightarrow$$
$$[\{(E\ x_3)(y_3)Q(x_2) \leftrightarrow Q(y_3)\} \leftrightarrow \{(E\ x_4)P(x_4) \leftrightarrow (y_4)Q(y_4)\}].$$

P. Andrews posed this problem at the Fourth Workshop on Automated Deduction, Austin, Texas, February 1979. He added that he was willing to send the first 500 clauses for free. Resolution theorem provers are drowned as a result of the many clauses generated by the PC–CNF translator as a consequence of seven equivalences, which double the length of the formula each time. INSURER, heavily invoking INSTANCE, resulted in 169 successful instance recognitions, reducing the formula to TRUE.

7. *Comparison with Related Work*

The importance of recognizing independent subgoals in theorem proving has been demonstrated earlier by Ernst in [5]. His theorem prover had a two-level structure: one for the recognition of subgoals and the second, a conventional theorem prover, for dealing with nondecomposable subgoals. His first component works toward a so-called simplified miniscope format. He acknowledges that miniscope is not produced since, for instance, distribution of $\wedge$ over $\vee$ in the context of a universal quantifier is not attempted. Notable is his reluctance to rewrite blindly an implicative formula into a disjunction: "This is the reason that $\rightarrow$ is not replaced by its definition in terms of $\vee$ and $\sim$ except under special circumstances." We gather that to do chaining, he used special procedures that were sensitive to the syntactic implicative structure of formulas.

Mechanical proofs of the Andrews problem were reported in [8]. Attacking the problem head on with their resolution machine was impossible, since they report that straightforward translation into CNF produces 1024 clauses, mostly of length 8. A clever translator, TAMPR, produced only (sic) 86 clauses, and a refutation was produced upon clause 1052. They illustrate TAMPR as follows:

> TAMPR was directed to translate formulae of the form $\sim(P \leftrightarrow Q)$ to $(P \vee Q) \wedge (\sim P \vee \sim Q)$. This transformation avoids generating the two tautologies, $(P \vee \sim P)$ and $(Q \vee \sim Q)$, which will generally not be recognized as tautologies when P and Q are quantified formulas themselves. [8, p. 30]

This quotation shows clearly that their TAMPR program is a special case of our INSURER/INSTANCE modules. Their brute force resolution prover, which required only 106 seconds to generate the 1052 clauses, is an unbeatable workhorse.

8. *Future Investigations*

The results reported above suggest to us that a deductive "architecture" built up from deductive specialists is promising. Certainly, it is advisable to pursue this road first, with the restriction that the deductive components are algorithmic and thus always halting. Examples include: a model evaluator to decide whether a subgoal is hopeless (since it is not true in a model) [6]; an equality substitution simplifier, which replaces complex terms by equal but less complex terms; an **if–then–else** recognizer, which can split a problem into two subproblems of lesser complexity; and so forth. At a certain point, this algorithmic restriction should be abandoned. Then the realm of search is entered again, no longer on the *modus ponens* level, but with operators of greater scope: Check whether it is worthwhile to introduce an abbreviation for a recurring expression; apply key theorem α in situation β try to adapt the proof for a similar result in a less general theory; try to prove a more general result that can be expressed more concisely (and that is not falsified by any available model); resort to induction in a specific context; try to reinterpret the theory under consideration into other available theories; and so forth.

Somehow, we must deal with the phenomenon that at a certain stage in a theory, some previous result will be applied "automatically" when it can be applied. Thus, when a theory becomes activated, some theorems will become active in a "compiled format," as additional derivation rules. At the same time, we doubt that this "compilation" is an all-or-nothing matter; a theorem can gradually reach the status of being applied automatically (while this process can always be backtracked).

Frequently, it has been emphasized that something should be done with a newly found proof, that it should be the input for some kind of a learning component. Somehow, nobody has ever designed a procedure that could do something useful with the many mechanical proofs that have been generated in the last decades. But even when we refrain from starting a learning process, we still need a description of the proof (and also of its associated theorem) in order to do analogy reasoning. We suspect that the lack of a greater variety of deductive operators, which hampers our proving interesting theorems, is also responsible for the impossibility of making sense of obtained proofs.

When a larger collection of operators in a theory is available, an obvious step is to assign them priorities, automatically on the basis of performance or initially by "Acts of God"—hence, by programmers. Then it will be possible to generate (recursively, thereby introducing another dimension in which search is performed) skeleton proofs, to be refined in the next level of recursion. In [13] Sacerdoti has obtained convincing results with this technique in the realm of plan generation.

Yet, there is still a fair chance that the problem of mechanically proving difficult mathematical conjectures can advantageously be replaced by another problem: how to generate automatically (with respect to a given collection of definitions, axioms, lemmas, theorems, models, and similar theories) an interesting conjecture or concept to be defined. This capability, at least to some extent, might be essential for generating intermediate stepping stones for a really difficult theorem.

ACKNOWLEDGMENTS. Extensive comments by the reviewers and the critique by Richard Weyhrauch have contributed substantially to the presentation. Julie Tilton heroically streamlined my version of English.

REFERENCES

1. DE CHAMPEAUX, D. A theorem prover dating a semantic network. In *Proceedings of AISB/GI Conference* (Hamburg, West Germany). Univ. of Hamburg, Hamburg, 1978, pp. 82–92.
2. DE CHAMPEAUX, D. Sub-problem finder and instance checker, two cooperating preprocessors for theorem provers. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence* (Tokyo). 1979, pp. 191–196.
3. CHANG, C -L., AND LEE, R. C. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, Orlando, Fla., 1973.
4. ENDERTON, H. B. *A Mathematical Introduction to Logic.* Academic Press, Orlando, Fla., 1972.
5. ERNST, G. W. The utility of independent subgoals in theorem proving. *Inf. Control 18* (1971), 237–252.
6. GELERNTER, H. Realization of a geometry theorem proving machine. In *Proceedings of an International Conference on Information Processing* (Paris), UNESCO House, 1959, pp. 273–282. Reprinted in *Computers and Thought,* E. A. Feigenbaum and J. Feldman, Ed. McGraw-Hill, New York, 1963, pp. 134–152.
7. GREEN, C. The application of theorem-proving to question-answering systems. Stanford Artificial Intelligence Project Memo AI-96, Stanford Univ., Stanford, Calif., June 1969.
8. HENSCHEN, L., ET AL. Challenge problem 1. *SIGART Newsl. 72* (July 1980), 30–31.
9. KLING, R. E. A paradigm for reasoning by analogy. *Artif. Intell. 2* (1971), 147–178.
10. LOVELAND, D. W. *Automated Theorem Proving: A Logical Basis.* North-Holland, The Netherlands, 1978.
11. MANNA, Z. *Introduction to Mathematical Theory of Computation.* McGraw-Hill, New York, 1972.
12. ROBINSON, J. A. A machine-oriented logic based on the resolution principle. *J. ACM 12,* 1, (Jan. 1965), 23–41.
13. SACERDOTI, E. D. Planning in a Hierarchy of Abstraction Spaces. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Stanford). 1973, pp. 412–422.
14. WANG, H. Toward mechanical mathematics. *IBM J* (Jan 1960), 2–22.

RECEIVED FEBRUARY 1984; REVISED NOVEMBER 1985; ACCEPTED DECEMBER 1985