10-1991

Resolution without Unification

William C. Purdy
Syracuse University, wcpurdy@ecs.syr.edu

SU-CIS-91-31

Resolution without Unification

William C. Purdy

October 1991

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, New York 13244-4100

Resolution without Unification

William C. Purdy*

October 1991

*School of Computer and Information Science, Syracuse University

Abstract

Resolution as an inference procedure forms the basis of most automated theorem-proving and reasoning systems. The most costly constituent of the resolution procedure in its conventional form is unification. This paper describes PCS, a first-order language in which resolution-based inference can be conducted without unification. PCS resembles the language of elementary logic with the difference that singular predicates supplant individual constants and functions. The result is a uniformity in the treatment of individual constants, functions and predicates. An especially costly part of unification is the occur check. Since unification is unnecessary for resolution in PCS, the occur check is completely circumvented. The conditions that would invoke an occur check are properly represented however. In this sense, resolution in PCS can be viewed as a refinement of conventional resolution. PCS does not have an identity relation. Nonetheless, identity can be expressed in PCS and deduction with identicals can be performed.

1 Introduction

Resolution [8] as an inference procedure forms the basis of most automated theorem-proving and reasoning systems. In its conventional form, resolution involves computation of a substitution, which is applied to the expressions to be resolved. This computation, called *unification*, is the most costly constituent of the resolution procedure, and so has been the subject of intensive study.

This paper describes PCS, a first-order language in which resolution-based inference can be conducted without unification. PCS resembles the language of elementary logic with the following difference. Singular predicates supplant individual constants and functions. It is not unusual to treat individual constants as nullary functions, nor to treat n-ary functions as $(n+1)$-ary predicates. But it appears that these devices have not been used together. When they are, the result is a uniformity in the treatment of individual constants, functions and predicates.

A computationally costly part of unification is the operation known as the *occur check*. The occur check prevents cyclic substitutions. Because of its high cost, it is simply ignored in most automated reasoning systems. For most applications, this causes no problem. However, soundness of the inference procedure is sacrificed. Computation with cyclic (infinite) terms has been investigated [4] to avoid the occur check while retaining soundness. Since unification is unnecessary for resolution in PCS, the occur check is completely circumvented. The conditions that would invoke an occur check are properly represented however. In this sense, resolution in PCS can be viewed as a refinement of conventional resolution.

PCS does not have an identity relation. Nonetheless, identity of singular expressions, which correspond to terms in conventional predicate calculus, can be expressed. Deduction with identicals can be performed in PCS using the same resolution-based inference procedure.

In the following sections, the syntax and semantics of PCS are defined. Then some special properties of singular expressions are established. Next the transformation to clausal form in PCS is described. Resolution in PCS is defined and shown to be sound and complete as a refutation procedure. The occur check is discussed in relation to resolution in PCS. Finally, deduction with identicals is introduced. Examples are presented to illustrate refutation in PCS. The treatment throughout is semantic; however, an axiomatic treatment can also be given (see [6]).

2 Definition of PCS

2.1 Syntax

The vocabulary of PCS consists of the following.

1. Predicate symbols $\mathcal{P}$ of two kinds (let $\omega_+ := \omega - \{0\}$):
   (a) ordinary predicate symbols $\mathcal{R} = (\bigcup_{n\in\omega_+} \mathcal{R}_n)$ where $\mathcal{R}_n = \{R_i^n : i \in \omega\}$, and
   (b) singular predicate symbols $\mathcal{S} = (\bigcup_{n\in\omega_+} \mathcal{S}_n)$ where $\mathcal{S}_n = \{S_i^n : i \in \omega\}$
2. Variable symbols $\mathcal{V} = \{v_i : i \in \omega\}$
3. Boolean operators $\wedge$ and $\neg$
4. Quantifier $\exists$
5. Parentheses (and)
6. Comma ,

There are no terms in PCS. In their stead, singular expressions are used. These are defined as follows:

1. if $S^1 \in \mathcal{S}_1$ and and $x \in \mathcal{V}$ then $S^1(x)$ is a singular expression
2. if $S^{n+1} \in \mathcal{S}_{n+1}$, $x, x_1, \ldots, x_n \in \mathcal{V}$ are distinct and $S_1, \ldots, S_n$ are singular expressions, then $\exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge S^{n+1}(x_1, \ldots, x_n, x)) \cdots)$ is a singular expression

3. nothing else is a singular expression

Expressions are defined as follows:

1. if $P^n \in (\mathcal{R}_n \cup \mathcal{S}_n)$ and $x_1, \ldots, x_n \in \mathcal{V}$, then $P^n(x_1, \ldots, x_n)$ is an expression
2. if ϕ is an expression then $\neg\phi$ is an expression
3. if ϕ, ψ are expressions then $(\phi \wedge \psi)$ is an expression
4. if ϕ is an expression and $x \in \mathcal{V}$ occurs free in ϕ, then $\exists x \phi$ is an expression
5. nothing else is an expression

Free and bound variables are defined in the usual way. When a list of variable symbols follows an expression symbol, e.g., $\phi(x_1, \ldots, x_n)$, these variables are all the free variables and only free variables in the expression. When the expression symbol is used without a list of variable symbols, it is left open which variables are free in that expression. As a general rule, it is assumed that all expressions are rectified.

Since the intended interpretation of $\exists x \phi(x_1, \ldots, x_n, x)$ is identical to that of $\exists y \phi(x_1, \ldots, x_n, y)$, PCS expressions are defined to be equivalence classes, each equivalence class consisting of all alphabetic variants. This equivalence can be defined formally (e.g., see Barnes and Mack [2]), but this will not be done here. Any member of a given equivalence class will be used to represent the class. Hence the two forms given above represent the same PCS expression.

In the sequel, parentheses are dropped whenever no confusion can result. Metavariables are used as follows: R^n ranges over $\mathcal{R}_n$; S^n ranges over $\mathcal{S}_n$; P^n ranges over $\mathcal{R}_n \cup \mathcal{S}_n$; x, y, z range over $\mathcal{V}$; S ranges over singular expressions; and ϕ, ψ, θ range over expressions. Applying subscripts to these symbols does not change their ranges.

2.2 **Semantics** An *interpretation* of PCS is a pair $\mathcal{I} = \langle \mathcal{D}, \mathcal{F} \rangle$ where $\mathcal{D}$ is a nonempty set and $\mathcal{F}$ is a mapping defined on $\mathcal{P}$ satisfying:

1. if $R^n \in \mathcal{R}_n$, then $\mathcal{F}(R^n) \subseteq \mathcal{D}^n$
2. if $S^{n+1} \in \mathcal{S}_{n+1}$, then $\mathcal{F}(S^{n+1}) \subseteq \mathcal{D}^{n+1}$ such that for all $d_1, \ldots, d_n \in \mathcal{D}$ there exists $d \in \mathcal{D}$ with $\langle d_1, \ldots, d_n, d \rangle \in \mathcal{F}(S^{n+1})$ and for all $d' \in \mathcal{D}$, $\langle d_1, \ldots, d_n, d' \rangle \in \mathcal{F}(S^{n+1})$ implies $d' = d$

Let $d_1, \ldots, d_n \in \mathcal{D}$ and $\phi(x_1, \ldots, x_n)$ be an expression of PCS. Then $\phi(x_1, \ldots, x_n)$ *is satisfied by* $d_1, \ldots, d_n$ *in* $\mathcal{I}$ (written $\mathcal{I} \models \phi[d_1, \ldots, d_n]$) iff one of the following holds:

1. $\phi(x_1, \ldots, x_n) = P^n(x_1, \ldots, x_n)$ and $\langle d_1, \ldots, d_n \rangle \in \mathcal{F}(P^n)$
2. $\phi(x_1, \ldots, x_n) = \neg\psi(x_1, \ldots, x_n)$ and $\mathcal{I} \not\models \psi[d_1, \ldots, d_n]$
3. $\phi(x_1, \ldots, x_n) = \psi(x_{i_1}, \ldots, x_{i_l}) \wedge \theta(x_{j_1}, \ldots, x_{j_m})$ and ($\mathcal{I} \models \psi[d_{i_1}, \ldots, d_{i_l}]$ and $\mathcal{I} \models \theta[d_{j_1}, \ldots, d_{j_m}]$), where $\{i_1, \ldots, i_l\} \cup \{j_1, \ldots, j_m\} = \{1, \ldots, n\}$
4. $\phi(x_1, \ldots, x_n) = \exists x \psi(x_1, \ldots, x_n, x)$ and there exists $d \in \mathcal{D}$ such that $\mathcal{I} \models \psi[d_1, \ldots, d_n, d]$

An expression $\phi(x_1,\ldots,x_n)$ is *true in* $\mathcal{I}$, written $\mathcal{I} \models \phi(x_1,\ldots,x_n)$, iff for all $d_1,\ldots,d_n \in \mathcal{D}$, $\mathcal{I} \models \phi[d_1,\ldots,d_n]$. $\phi(x_1,\ldots,x_n)$ is *valid*, written $\models \phi(x_1,\ldots,x_n)$, iff $\phi(x_1,\ldots,x_n)$ is true in every interpretation.

2.3 Abbreviations

PCS is extended by the following abbreviations.

1. $\psi \vee \theta := \neg(\neg\psi \wedge \neg\theta)$

2. $\psi \rightarrow \theta := \neg(\psi \wedge \neg\theta)$

3. $\psi \leftrightarrow \theta := (\psi \rightarrow \theta) \wedge (\theta \rightarrow \psi)$

4. $\forall x\psi := \neg\exists x\neg\psi$

The semantics for these abbreviations can be given directly as follows:

1. If $\phi(x_1,\ldots,x_n) = \psi(x_{i_1},\ldots,x_{i_l}) \vee \theta(x_{j_1},\ldots,x_{j_m})$ then $\mathcal{I} \models \phi[d_1,\ldots,d_n]$ iff ($\mathcal{I} \models \psi[d_{i_1},\ldots,d_{i_l}]$ or $\mathcal{I} \models \theta[d_{j_1},\ldots,d_{j_m}]$), where $\{i_1,\ldots,i_l\} \cup \{j_1,\ldots,j_m\} = \{1,\ldots,n\}$

2. If $\phi(x_1,\ldots,x_n) = \psi(x_{i_1},\ldots,x_{i_l}) \rightarrow \theta(x_{j_1},\ldots,x_{j_m})$ then $\mathcal{I} \models \phi[d_1,\ldots,d_n]$ iff ($\mathcal{I} \models \psi[d_{i_1},\ldots,d_{i_l}]$ implies $\mathcal{I} \models \theta[d_{j_1},\ldots,d_{j_m}]$), where $\{i_1,\ldots,i_l\} \cup \{j_1,\ldots,j_m\} = \{1,\ldots,n\}$

3. If $\phi(x_1,\ldots,x_n) = \psi(x_{i_1},\ldots,x_{i_l}) \leftrightarrow \theta(x_{j_1},\ldots,x_{j_m})$ then $\mathcal{I} \models \phi[d_1,\ldots,d_n]$ iff ($\mathcal{I} \models \psi[d_{i_1},\ldots,d_{i_l}]$ iff $\mathcal{I} \models \theta[d_{j_1},\ldots,d_{j_m}]$), where $\{i_1,\ldots,i_l\} \cup \{j_1,\ldots,j_m\} = \{1,\ldots,n\}$

4. If $\phi(x_1,\ldots,x_n) = \forall x \psi(x_1,\ldots,x_n,x)$ then $\mathcal{I} \models \phi[d_1,\ldots,d_n]$ iff for all $d \in \mathcal{D}$, $\mathcal{I} \models \psi[d_1,\ldots,d_n,d]$

From the definition of truth in $\mathcal{I}$, it follows that $\mathcal{I} \models \phi(x_1,\ldots,x_n)$ iff $\mathcal{I} \models \forall x_1 \cdots \forall x_n \phi(x_1,\ldots,x_n)$. Clearly this holds for every universal closure of $\phi(x_1,\ldots,x_n)$ (i.e., every permutation of the prefix $\forall x_1 \cdots \forall x_n$). That every universal closure of ϕ is true in $\mathcal{I}$ will be written $\mathcal{I} \models \forall\phi$.

Two useful lemmas follow directly from these remarks.

LEMMA 1 *If $\mathcal{I} \models \forall(\phi \rightarrow \psi)$, then $\mathcal{I} \models \forall\phi$ implies $\mathcal{I} \models \forall\psi$.*

LEMMA 2 *If τ is obtained from a Boolean tautology by uniform substitution of PCS expressions for propositional variables, then $\models \forall\tau$.*

3 Properties of singular expressions

Singular expressions play a central role in PCS. The denotation of a singular expression is a single (though not necessarily unique) individual. Singular expressions commute in a certain way with the Boolean operators. These properties are established in this section.

LEMMA 3 *There exists $d \in \mathcal{D}$ such that $\mathcal{I} \models S[d]$ and for all $d' \in \mathcal{D}$, $\mathcal{I} \models S[d']$ implies $d' = d$.*

proof: Define the depth of a singular expression as follows. $depth(S^1(x)) := 0$. $depth(\exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge S^{n+1}(x_1, \ldots, x_n, x)) \cdots)) := 1 + max\{depth(S_i(x_i)) : 1 \leq i \leq n\}$. The proof is a straightforward induction on the depth of $S(x)$.

In the following, Lemma 3 will be abbreviated $\exists! d \in \mathcal{D} : \mathcal{I} \models S[d]$.

LEMMA 4 *$\mathcal{I} \models \exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge \neg\phi(x_1, \ldots, x_n)) \cdots)$ iff $\mathcal{I} \models \neg\exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge \phi(x_1, \ldots, x_n)) \cdots)$.*

proof: $\mathcal{I} \models \exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge \neg\phi(x_1, \ldots, x_n)) \cdots)$ iff $\exists! d_1 \cdots \exists! d_n : (\mathcal{I} \models S_1[d_1]) \wedge \cdots \wedge (\mathcal{I} \models S_n[d_n]) \wedge (\mathcal{I} \models \neg\phi[d_1, \ldots, d_n])$ iff $\exists! d_1 \cdots \exists! d_n : (\mathcal{I} \models S_1[d_1]) \wedge \cdots \wedge (\mathcal{I} \models S_n[d_n]) \wedge (\mathcal{I} \not\models \phi[d_1, \ldots, d_n])$ iff $\mathcal{I} \not\models \exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge \phi(x_1, \ldots, x_n)) \cdots)$ iff $\mathcal{I} \models \neg\exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge \phi(x_1, \ldots, x_n)) \cdots)$ (follows from the definition of satisfaction and Lemma 3).

COROLLARY 5 *$\mathcal{I} \models \exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge \phi(x_1, \ldots, x_n)) \cdots)$ iff $\mathcal{I} \models \forall x_1(S_1(x_1) \rightarrow \cdots \rightarrow \forall x_n(S_n(x_n) \rightarrow \phi(x_1, \ldots, x_n)) \cdots)$.*

LEMMA 6 *$\mathcal{I} \models \exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge \phi(x_{i_1}, \ldots, x_{i_l}) \wedge \psi(x_{j_1}, \ldots, x_{j_m})) \cdots)$ iff $(\mathcal{I} \models \exists x_{i_1}(S_{i_1}(x_{i_1}) \wedge \cdots \wedge \exists x_{i_l}(S_{i_l}(x_{i_l}) \wedge \phi(x_{i_1}, \ldots, x_{i_l})) \cdots)$ and $\mathcal{I} \models \exists x_{j_1}(S_{j_1}(x_{j_1}) \wedge \cdots \wedge \exists x_{j_m}(S_{j_m}(x_{j_m}) \wedge \psi(x_{j_1}, \ldots, x_{j_m})) \cdots))$, where $\{i_1, \ldots, i_l\} \cup \{j_1, \ldots, j_m\} = \{1, \ldots, n\}$.*

proof: $\mathcal{I} \models \exists x_1(S_1(x_1) \wedge \cdots \wedge \exists x_n(S_n(x_n) \wedge \phi(x_{i_1}, \ldots, x_{i_l}) \wedge \psi(x_{j_1}, \ldots, x_{j_m})) \cdots)$ iff $\exists! d_1 \cdots \exists! d_n : (\mathcal{I} \models S_1[d_1]) \wedge \cdots \wedge (\mathcal{I} \models S_n[d_n]) \wedge (\mathcal{I} \models (\phi(x_{i_1}, \ldots, x_{i_l}) \wedge \psi(x_{j_1}, \ldots, x_{j_m})$ $[d_{j_1}, \ldots, d_{j_m}])$ iff $\exists! d_1 \cdots \exists! d_n : (\mathcal{I} \models S_1[d_1]) \wedge \cdots \wedge (\mathcal{I} \models S_n[d_n]) \wedge (\mathcal{I} \models \phi[d_{i_1}, \ldots, d_{i_l}]) \wedge$ $(\mathcal{I} \models \psi[d_{j_1}, \ldots, d_{j_m}])$ iff $(\mathcal{I} \models \exists x_{i_1}(S_{i_1}(x_{i_1}) \wedge \cdots \wedge \exists x_{i_l}(S_{i_l}(x_{i_l}) \wedge \phi(x_{i_1}, \ldots, x_{i_l})) \cdots))$ $\wedge(\mathcal{I} \models \exists x_{j_1}(S_{j_1}(x_{j_1}) \wedge \cdots \wedge \exists x_{j_m}(S_{j_m}(x_{j_m}) \wedge \psi(x_{j_1}, \ldots, x_{j_m})) \cdots))$ (follows from the definition of satisfaction and Lemma 3).

LEMMA 7 *$\mathcal{I} \models \forall(S_i^{n+1}(x_1, \ldots, x_n, x) \rightarrow (S_j^{m+1}(y_1, \ldots, y_m, x) \vee \phi))$ iff $\mathcal{I} \models \forall(S_j^{m+1}(y_1, \ldots, y_m, x) \rightarrow (S_i^{n+1}(x_1, \ldots, x_n, x) \vee \phi))$, providing x is not free in ϕ and is distinct from $x_1, \ldots, x_n, y_1, \ldots, y_m$.*

proof: Let $\alpha : \mathcal{V} \rightarrow \mathcal{D}$ be an assignment to variables, and let $\mathcal{I} \models S_i^{n+1}[\alpha]$ be an abbreviation for $\mathcal{I} \models S_i^{n+1}[\alpha(x_1), \ldots, \alpha(x_n), \alpha(x)]$. Then it follows from the definition of satisfaction that $\mathcal{I} \models \forall(S_i^{n+1}(x_1, \ldots, x_n, x) \rightarrow (S_j^{m+1}(y_1, \ldots, y_m, x) \vee \phi))$ iff for each α, $\mathcal{I} \models S_i^{n+1}[\alpha]$ implies either $\mathcal{I} \models S_j^{m+1}[\alpha]$ or $\mathcal{I} \models \phi[\alpha]$. If $\mathcal{I} \models \phi[\alpha]$, the lemma follows. Suppose then that $\mathcal{I} \not\models \phi[\alpha]$. Since x is not free in ϕ and is distinct from $x_1, \ldots, x_n, y_1, \ldots, y_m$, there is an assignment α' that agrees with α off x such that $\mathcal{I} \models S_i^{n+1}[\alpha']$. In this case, $\mathcal{I} \models S_j^{m+1}[\alpha']$ as well. But by Lemma 3, α' is the *only* assignment that agrees with α off x having this property. Hence $\mathcal{I} \models S_i^{n+1}[\alpha']$ iff $\mathcal{I} \models S_j^{m+1}[\alpha']$. This completes the proof.

4 **Skolem form** An expression is in *prenex form* iff it is an instance of the schema $Q_1 \cdots Q_n M$, where $0 \leq n$, each Q_i is either $\exists x_i$ or $\forall x_i$, and M is an expression containing no occurrences of $\exists$ or $\forall$. $Q_1 \cdots Q_n$ is the *prefix* and M is the *matrix* of the expression. Given any closed expression, construction of a corresponding prenex form in PCS and proof of their logical equivalence is the same as for conventional predicate calculus (e.g., see Enderton [5]).

LEMMA 8 *For every closed expression there exists a logically equivalent prenex form expression.*

Let $\phi = Q_1 \cdots Q_n M$ be a prenex form expression. Then $\star\phi$, its corresponding *Skolem form*, can be constructed in PCS. First, for $0 \leq k$, $\star^k\phi$ is defined inductively as follows. $\star^0\phi := \phi$. If $\star^k\phi = \forall x_1 \cdots \forall x_m \exists x_{m+1} Q_{m+2} \cdots Q_n M$, where $0 \leq k$ and $0 \leq m$, then $\star^{k+1}\phi = \forall x_1 \cdots \forall x_m \forall x_{m+1} Q_{m+2} \cdots Q_n (S^{m+1}(x_1, \ldots, x_m, x_{m+1}) \rightarrow M)$, where S^{m+1} is a singular predicate symbol that has no previous occurrence. This defines a construction. Now, $\star\phi := \star^q\phi$, where q is the number of existential quantifiers in the prefix of ϕ.

LEMMA 9 *For every closed expression ϕ there exists a Skolem form $\star\phi$ such that $\star\phi$ is satisfiable iff ϕ is satisfiable.*

proof: It may be assumed that ϕ is in prenex form. It suffices to prove that $\star^k\phi$ is satisfiable iff $\star^{k+1}\phi$ is satisfiable. Let $\star^k\phi = \forall x_1 \cdots \forall x_m \exists x_{m+1} Q_{m+2} \cdots Q_n M$, where $0 \leq k$ and $0 \leq m$ and $\star^{k+1}\phi = \forall x_1 \cdots \forall x_m \forall x_{m+1} Q_{m+2} \cdots Q_n (S^{m+1}(x_1, \ldots, x_m, x_{m+1}) \rightarrow$

M). Suppose $\mathcal{I} \models \star^k \phi$. Then $\forall d_1 \cdots \forall d_m \exists d_{m+1} : \mathcal{I} \models Q_{m+2} \cdots Q_n M[d_1, \ldots, d_m, d_{m+1}]$. Since the denotation of S^{m+1} is irrelevant for satisfaction of $\star^k \phi$ in $\mathcal{I}$, let $\mathcal{I}'$ be an interpretation like $\mathcal{I}$ except that $\mathcal{F}'(S^{m+1}) = \{\langle d_1, \ldots, d_m, d_{m+1} \rangle : \varepsilon d_{m+1}(\mathcal{I} \models Q_{m+2} \cdots Q_n M\ [d_1, \ldots, d_m, d_{m+1}])\}$, where ε is a choice function. For a denumerable domain, d_{m+1} can be specified as the first element in an enumeration of $\mathcal{D}$ that satisfies $\mathcal{I} \models Q_{m+2} \cdots Q_n M\ [d_1, \ldots, d_m, d_{m+1}]$, thus eliminating the need for ε. With this definition for S^{m+1}, $\mathcal{I}' \models \star^{k+1} \phi$. Conversely, if $\mathcal{I} \models \star^{k+1} \phi$, then by the definition of an interpretation, $\forall d_1 \cdots \forall d_m \exists d_{m+1} : \mathcal{I} \models S^{m+1}[d_1, \ldots, d_m, d_{m+1}]$. Thence, by the definition of satisfaction, $\mathcal{I} \models \star^k \phi$.

5 **Clausal form** An *atom* is an n-ary predicate symbol followed by a list of n variables, e.g., $P^n(x_1, \ldots, x_n)$. A *literal* is an atom or an atom with a prefixed complement operator. If the literal contains a complement operator, it is *negative*, otherwise *positive*. An atom or literal is singular if the predicate symbol is singular. A *clause* is the universal closure of a finite disjunction of literals. The common practice of writing a clause without the quantifier prefix will be followed. Also a clause will sometimes be written as a set of literals. Which form is being used will be clear from the context. This implies that a clause is actually an equivalence class where the equivalence is defined by the associative, commutative and idempotent properties of disjunction. The *empty clause*, consisting of no literals, is written $\square$. A *clausal form* is a finite conjunction of clauses.

The following lemma follows from the existence of Skolem form and negation normal form for every closed expression.

LEMMA 10 *For every closed expression ϕ there exists a clausal form D such that ϕ is satisfiable iff D is satisfiable.*

A variable occurrence as the rightmost argument of a singular predicate is a *singular occurrence*; other occurrences are *nonsingular*. A variable that has a singular occurrence in a negative literal is *constrained*, otherwise, unconstrained. If a variable has only one occurrence in a clause and that occurrence is in a negative singular literal, then the literal in which it occurs is *improper*. A clause is *proper* if it has no improper literals.

LEMMA 11 *If C is a clause and L an improper literal in C, then $C - \{L\}$ and C are logically equivalent.*

proof: Let $C(x_1, \ldots, x_n, x) = L(x_{i_1}, \ldots, x_{i_l}, x) \vee C'(x_{j_1}, \ldots, x_{j_m})$, where $\{i_1, \ldots, i_l\} \cup \{j_1, \ldots, j_m\} = \{1, \ldots, n\}$. $\mathcal{I} \models \forall(L(x_{i_1}, \ldots, x_{i_l}, x) \vee C'(x_{j_1}, \ldots, x_{j_m}))$ iff for all $d_1, \ldots, d_n \in \mathcal{D} : \mathcal{I} \models \forall x(L(x_{i_1}, \ldots, x_{i_l}, x) \vee C'(x_{j_1}, \ldots, x_{j_m}))[d_1, \ldots, d_n]$ iff for all $d_1, \ldots, d_n \in \mathcal{D}$: (for all $d \in \mathcal{D}$: $\mathcal{I} \models L[d_{i_1}, \ldots, d_{i_l}, d]$) or $\mathcal{I} \models C'[d_{j_1}, \ldots, d_{j_m}]$. But $L(x_{i_1}, \ldots, x_{i_l}, x) = \neg S^{l+1}(x_{i_1}, \ldots, x_{i_l}, x)$. Therefore, (for all $d \in \mathcal{D}$: $\mathcal{I} \models L[d_{i_1}, \ldots, d_{i_l}, d]$) cannot hold in $\mathcal{I}$ for *any* $d_{i_1}, \ldots, d_{i_l} \in \mathcal{D}$ since $\exists! d : \mathcal{I} \models S[d_{i_1}, \ldots, d_{i_l}, d]$. Hence for all $d_{j_1}, \ldots, d_{j_m} \in \mathcal{D} : \mathcal{I} \models C'[d_{j_1}, \ldots, d_{j_m}]$, i.e., $\mathcal{I} \models \forall C'(x_{j_1}, \ldots, x_{j_m})$.

If C is a clause, $var(C)$ is the set of variables occurring in C. A *substitution* is a mapping $\sigma : \mathcal{V} \rightarrow \mathcal{V}$. $C\sigma$ will denote the clause obtained from clause C by applying substitution σ to each of the variable occurrences in C. Since all variable occurrences are bound, any substitution that is bijective will yield the same clause (an alphabetic variant). Substitutions are closed under composition.

If C is a clause and σ a substitution, then $C' = C\sigma$ is a *factor* of C. If C' contains fewer literals than C, then C' is a *proper factor*. A proper factor is formed when σ is not bijective and makes previously distinct literals identical. The following lemma is immediate from the definition of satisfaction and truth.

LEMMA 12 *If C is a clause and C' is a factor of C, then C' is a logical consequence of C.*

Let C_1 and C_2 be clauses and σ be a substitution. If $C_1\sigma \subseteq C_2$, then C_1 *subsumes* C_2. From Lemmas 1 and 2 and the Boolean tautology $p \rightarrow (p \vee q)$, it follows that if $C_1\sigma \subseteq C_2$, then $\forall C_2$ is a logical consequence of $\forall C_1\sigma$. Hence by Lemma 12, if C_1 subsumes C_2, then $\forall C_2$ is a logical consequence of $\forall C_1$.

If C is a clause and S is a singular atom, then $C' = \neg S \vee C$ is an *instance* of C. C' is a *proper instance* if some $x \in var(C)$ is unconstrained in C and constrained in C'. If C' is an instance (proper instance) of C, then an instance (proper instance) of C' is an instance (proper instance) of C. An instance with no unconstrained variables is a *ground instance*. Since an instance C' of C is subsumed by C, it follows that C' is a logical consequence of C.

6 Resolution in PCS

Let C_1 and C_2 be clauses containing literals $L_1 = P^n(x_1, \ldots, x_n)$ and $L_2 = \neg P^n(y_1, \ldots, y_n)$, respectively. Moreover, let $var(C_1)$ be disjoint from $var(C_2)$. Let σ be a substitution such that $\sigma(x_i) = \sigma(y_i)$ for $1 \leq i \leq n$. Then $(C_1 - \{L_1\})\sigma \cup (C_2 - \{L_2\})\sigma$ is a *resolvent* of C_1 and C_2. It is a *proper* resolvent if it is a proper clause.

Let D be a clausal form and C be a clause. A *deduction* of C from D, written $D \vdash C$, is a sequence $C_1, \ldots, C_n = C$ of clauses, where for $1 < i \leq n$, C_i is either a clause of D, or a proper resolvent of C_j and C_k for some $j, k < i$. A *refutation* of D is a deduction of the empty clause from D, written $D \vdash \Box$.

THEOREM 13 (Soundness of Resolution) *If C_1 and C_2 are clauses, then any resolvent of C_1 and C_2 is a logical consequence of $C_1 \wedge C_2$.*

proof: Let $C_1 = C_1' \vee P^n(x_1, \ldots, x_n)$ and $C_2 = C_2' \vee \neg P^n(y_1, \ldots, y_n)$. By Lemma 12, $C_1\sigma$ is a logical consequence of C_1 and $C_2\sigma$ is a logical consequence of C_2. Hence, using the definition of satisfaction, $C_1\sigma \wedge C_2\sigma$ is a logical consequence of $C_1 \wedge C_2$. Since $((p \vee q) \wedge (r \vee \neg q)) \rightarrow (p \vee r)$ is a Boolean tautology, Lemma 2 yields $\models \forall(((C_1'\sigma \vee P^n(z_1, \ldots, z_n)) \wedge (C_2'\sigma \vee \neg P^n(z_1, \ldots, z_n))) \rightarrow (C_1'\sigma \vee C_2'\sigma))$. Finally Lemma 1 gives the desired result.

It follows from this theorem that deduction as defined above is a sound procedure, i.e., $D \vdash C$ only if C is a logical consequence of D.

In view of the properties of singular expressions stated in Section 3, it is clear that a ground instance of a clause is logically equivalent to a Boolean combination of *atomic ground instances* of the form $\forall(\neg S_1(x_1) \vee \cdots \vee \neg S_n(x_n) \vee P^n(x_1, \ldots, x_n))$. Considering these atomic ground instances as prime expressions, the truth-functional properties of the Boolean expression may be investigated. The next lemma states that the deduction procedure defined above is complete for recognizing truth-functional contradiction.

LEMMA 14 (Ground Completeness) *If D is a ground clausal form, and D is a truth-functional contradiction, then $D \vdash \Box$.*

proof: A proof can be found in Andrews [1] (Theorem 1600).

Let D be a clausal form. The *lexicon* of D is the set of singular predicates occurring in D, with the provision that if no unary singular predicate occurs in D, S_0^1 is added to the lexicon. Ground instances of D formed using only elements of the lexicon will be called *Herbrand instances.* The conjunction of a finite number of Herbrand instances will be called a *compound* Herbrand instance (cH-instance).

LEMMA 15 (Herbrand's Theorem) *If D is a clausal form, and D is unsatisfiable, then some compound Herbrand instance of D is truth-functionally contradictory.*

proof: The proof is an adaptation of Andrews [1] Theorem 3503. Suppose that D has no truth-functionally contradictory cH-instances. Then there exists a truth-functional assignment $\mathcal{G}$ to atomic ground instances that validates all the cH-instances of D. (Here an appeal is made to the compactness of the propositional calculus - see Andrews [1] Theorem 1501.) $\mathcal{G}$ is now used to construct a model $\mathcal{M} = \langle \mathcal{D}, \mathcal{F} \rangle$ for D as follows. Let $\mathcal{D}$ be the set of all singular expressions constructed from the lexicon of D reduced by the equivalence $\approx$, defined as the least equivalence such that

(i) for any $x, y \in \mathcal{V} : S(x) \approx S(y)$

(ii) if $\mathcal{G}(\forall x_1 \cdots \forall x_n \forall x(\neg S_1(x_1) \vee \cdots \vee \neg S_n(x_n) \vee \neg S(x) \vee S^{n+1}(x_1, \ldots, x_n, x))) =$ **true** then $S(x) \approx \forall x_1 \cdots \forall x_n(\neg S_1(x_1) \vee \cdots \vee \neg S_n(x_n) \vee S^{n+1}(x_1, \ldots, x_n, x))$

(iii) if $S_i(x) \approx S_j(x)$ then $\forall x_1 \cdots \forall x_n(\neg S_1(x_1) \vee \cdots \vee \neg S_i(x_i) \vee \cdots \vee \neg S_n(x_n) \vee S^{n+1}(x_1, \ldots, x_n, x)) \approx \forall x_1 \cdots \forall x_n(\neg S_1(x_1) \vee \cdots \vee \neg S_j(x_i) \vee \cdots \vee \neg S_n(x_n) \vee S^{n+1} (x_1, \ldots, x_n, x))$

$\mathcal{D}$ corresponds to the *Herbrand universe* of D. In the following, let any singular expression represent its equivalence class.

$\mathcal{F}$ is defined:

(i) for each S^{n+1} in the lexicon of D, $\mathcal{F}(S^{n+1}) = \{\langle S_1, \ldots, S_n, \forall x_1 \cdots \forall x_n(\neg S_1(x_1) \vee \cdots \vee \neg S_n(x_n) \vee S^{n+1}(x_1, \ldots, x_n, x))\rangle : S_1, \ldots, S_n \in \mathcal{D}\}$

(ii) for each R^n occurring in D, $\mathcal{F}(R^n) = \{\langle S_1, \ldots, S_n\rangle : \mathcal{G}(\forall x_1 \cdots \forall x_n(\neg S_1(x_1) \vee \cdots \vee \neg S_n(x_n) \vee R^n(x_1, \ldots, x_n))) = \textbf{true}\}$

It follows immediately from (i) that for any singular expression $S(x)$, $\mathcal{I} \models S[S(x)]$.

Claim: If D is a clausal form that has no truth-functionally contradictory cH-instances, and $\mathcal{M}$ is defined as above, then $\mathcal{M} \models D$. Proof of the claim is by induction on $h =$ the number of unconstrained variables in D.

(i) $h = 0$. In this case, D is already a cH-instance, and therefore is validated by $\mathcal{G}$. Hence $\mathcal{M} \models D$.

(ii) $h > 0$. This case employs an embedded induction on the number of clauses in D.

(a) Suppose $D = \forall x \forall x_1 \cdots \forall x_n C$, where x is an unconstrained variable of D. Let $S(x)$ be an arbitrary singular expression in $\mathcal{D}$, and consider $D' = \forall x \forall x_1 \cdots \forall x_n(\neg S(x) \vee C)$. Every cH-instance of D' is also a cH-instance of D, which is validated by $\mathcal{G}$. Therefore by the inductive hypothesis, $\mathcal{M} \models \forall x \forall x_1 \cdots \forall x_n(\neg S(x) \vee C)$. But $\exists! d \in \mathcal{D} : \mathcal{M} \models S[d]$, viz., $d = S(x)$. Therefore, $\mathcal{M} \models \forall x_1 \cdots \forall x_n C[S(x)]$ for every singular expression $S(x) \in \mathcal{D}$. Hence by the definition of satisfaction, $\mathcal{M} \models D$.

(b) Suppose $D = D_1 \wedge D_2$. Let G_1 and G_2 be any cH-instances of D_1 and D_2, respectively. Then $G_1 \wedge G_2$ is a cH-instance of D, and so is validated by $\mathcal{G}$. But then G_1 and G_2 are validated by $\mathcal{G}$ as well. By the induction hypothesis $\mathcal{M} \models D_1$ and $\mathcal{M} \models D_2$. Hence by the definition of satisfaction, $\mathcal{M} \models D$.

LEMMA 16 (Lifting Lemma) *Let B_1 and B_2 be subsumed by clauses C_1 and C_2, respectively. If B is a resolvent of B_1 and B_2, then either (i) there exists a resolvent C of C_1 and C_2 such that B is subsumed by C or (ii) B is subsumed by C_1 or by C_2.*

proof: Let $B_1 = C_1\lambda \cup B_1'$ and $B_2 = C_2\lambda \cup B_2'$. Let $L_1 \in B_1$ and $L_2 \in B_2$ be the literals and σ the substitution involved in the resolution. Consider two cases.

(i) $L_1 \in C_1\lambda$ and $L_2 \in C_2\lambda$. Then $B = (B_1 - \{L_1\})\sigma \cup (B_2 - \{L_2\})\sigma = ((C_1\lambda \cup B_1') - \{L_1\})\sigma \cup ((C_2\lambda \cup B_2') - \{L_2\})\sigma = (C_1\lambda - \{L_1\})\sigma \cup (C_2\lambda - \{L_2\})\sigma \cup B_1'\sigma \cup B_2'\sigma$. Then $C = (C_1\lambda - \{L_1\})\sigma \cup (C_2\lambda - \{L_2\})\sigma$ is a resolvent of C_1 and C_2 that subsumes B. This argument is simplified by the assumption that none of the literals in B are improper. Actually some of the literals may drop out, but this does not alter the conclusion.

(ii) $L_1 \in B_1'$ or $L_2 \in B_2'$. Suppose $L_1 \in B_1'$. Then $B = (B_1 - \{L_1\})\sigma \cup (B_2 - \{L_2\})\sigma = ((C_1\lambda \cup B_1') - \{L_1\})\sigma \cup (B_2 - \{L_2\})\sigma = C_1\lambda\sigma \cup ((B_1' - \{L_1\}) \cup (B_2 - \{L_2\}))\sigma$. Therefore, $C_1\lambda\sigma \subseteq B$, i.e., B is subsumed by C_1. The argument is similar for $L_2 \in B_2'$.

THEOREM 17 (Completeness) *D is unsatisfiable only if $D \vdash \Box$.*

proof: If D is unsatisfiable, then by Herbrand's Theorem, there exists an unsatisfiable cH-instance G of D. By the completeness of ground deduction, $G \vdash \Box$. Let the refutation be $B_1, \ldots, B_n = \Box$. Construct a sequence of clauses $C_1, \ldots, C_n$, such that for $1 \leq i \leq n$, C_i subsumes B_i, as follows. Arguing inductively, assume that $C_1, \ldots, C_{i-1}$ has been constructed. If B_i is a clause of G, let C_i be the clause of D such that B_i is an instance of C_i. If B_i is a resolvent of B_j and B_k, then by the induction hypothesis, C_j and C_k subsume B_j and B_k, respectively. Now by the Lifting Lemma, B_i is subsumed by either a resolvent C of C_j and C_k, or by C_j, or by C_k. Choose the appropriate one for C_i. Thus $C_1, \ldots, C_n$ is a deduction. Since C_n subsumes $B_n = \Box$, this sequence is a refutation. Therefore $D \vdash \Box$.

The following example, taken from Chang and Lee [3] (p. 89), illustrates resolution in PCS.

The premises are:

$$\forall x((E(x) \land \neg V(x)) \rightarrow \exists y(S(x,y) \land C(y)))$$

$$\exists x(P(x) \land E(x) \land \forall y(S(x,y) \rightarrow P(y)))$$

$$\forall x(P(x) \rightarrow \neg V(x))$$

and the conclusion is:

$$\exists x(P(x) \land C(x))$$

In Skolem form (with the conclusion denied):

$$\forall x \forall y(S_0^2(x,y) \rightarrow ((E(x) \land \neg V(x)) \rightarrow (S(x,y) \land C(y))))$$

$$\forall x \forall y(S_0^1(x) \rightarrow (P(x) \land E(x) \land (S(x,y) \rightarrow P(y))))$$

$$\forall x(P(x) \rightarrow \neg V(x))$$

$$\forall x(P(x) \rightarrow \neg C(x))$$

where S_0^1 and S_0^2 are singular predicates.

In clausal form:

1. $\neg S_0^2(x,y) \lor \neg E(x) \lor V(x) \lor S(x,y)$

2. $\neg S_0^2(x,y) \vee \neg E(x) \vee V(x) \vee C(y)$

3. $\neg S_0^1(x) \vee P(x)$

4. $\neg S_0^1(x) \vee E(x)$

5. $\neg S_0^1(x) \vee \neg S(x,y) \vee P(y)$

6. $\neg P(x) \vee \neg V(x)$

7. $\neg P(x) \vee \neg C(x)$

The following sequence of clauses, appended to the premises, is a refutation. The justification for each clause is given in parentheses.

8. $\neg S_0^1(x) \vee \neg V(x)$ (resolve 3,6)

9. $\neg S_0^1(x) \vee \neg S_0^2(x,y) \vee V(x) \vee C(y)$ (resolve 2,4)

10. $\neg S_0^1(x) \vee \neg S_0^2(x,y) \vee C(y)$ (resolve 8,9)

11. $\neg S_0^1(x) \vee \neg S_0^2(x,y) \vee V(x) \vee S(x,y)$ (resolve 1,4)

12. $\neg S_0^1(x) \vee \neg S_0^2(x,y) \vee S(x,y)$ (resolve 8,11)

13. $\neg S_0^1(x) \vee \neg S_0^2(x,y) \vee P(y)$ (resolve 5,12)

14. $\neg S_0^1(x) \vee \neg S_0^2(x,y) \vee \neg C(y)$ (resolve 7,13)

15. $\square$ (resolve 10,14)

7 The occur check Resolution in PCS does not involve unification, and so it does not involve an occur check either. However, it is of interest to examine those situations in which an occur check would inhibit resolution in conventional predicate calculus.

A simple example often used is the following.

1. $\neg P(z, z)$

2. $P(x, f(x))$

Here an occur check blocks unification. But if the occur check is ignored, $\Box$ is erroneously deduced.

In PCS this example is represented as follows.

1. $\neg P(z, z)$

2. $\neg S_f^2(x, y) \vee P(x, y)$

Using the substitution $[z/x, z/y]$, the resolvent is $\neg S_f^2(z, z)$. Since this is a proper clause and no further resolution is possible, a refutation is not obtained. The resolvent asserts $\forall z \neg S_f^2(z, z)$, or equivalently, $\neg \exists z S_f^2(z, z)$. That is, the singular predicate S_f^2 is irreflexive. This is equivalent to asserting that the corresponding function has no fixed-point. If it were given that at least one fixed-point exists, i.e., $\exists z S_f^2(z, z)$, or in clausal form, $\neg S_{fp}^1(v) \vee S_f^2(v, v)$, then a refutation would follow.

Thus resolution in PCS is a refinement of resolution in conventional predicate calculus. It may also be compared with the extension of conventional predicate calculus to cyclic terms. $\exists z S_f^2(z,z)$ asserts the existence of a value for the cyclic term $f(f(f(\cdots)))$.

8 Identity in PCS PCS does not have an identity relation. Nonetheless, identity of singular expressions can be expressed. $\exists x(S_1(x) \wedge S_2(x))$, or equivalently $\forall x(S_1(x) \rightarrow S_2(x))$, expresses the identity of singular expressions S_1 and S_2. The only deficit relative to predicate calculus with a logical identity relation is the inability to express $x = y$ (see [7]). Indeed, except for expressions of this form, translation between PCS and predicate calculus with identity (PCI) can be accomplished by means of the correspondence:

$S_f^{n+1}(x_1, \ldots, x_n, x)$ in PCS corresponds to $f^n(x_1, \ldots, x_n) = x$ in PCI.

This places the expressiveness of PCS properly between that of predicate calculus without identity and predicate calculus with identity.

Reasoning in PCS with identicals needs only resolution and the *rule of symmetrical pairs.* This rule, justified by Lemma 7, is the following.

> Let clause $C = \neg S_i^{n+1}(x_1, \ldots, x_n, x) \vee S_j^{m+1}(y_1, \ldots, y_m, x) \vee B$, where $x \notin \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_m\} \cup var(B)$. Then from C infer $C' = S_i^{n+1}(x_1, \ldots, x_n, x) \vee \neg S_j^{m+1}(y_1, \ldots, y_m, x) \vee B$.

$S_i^{n+1}(x_1, \ldots, x_n, x)$ and $S_j^{m+1}(y_1, \ldots, y_m, x)$ are called a *symmetrical pair.* This rule provides for the substitutivity of identicals. It might be argued that it does so more simply than either the rule of substitution or the functional reflexive axioms together with the rule of paramodulation.

In the sequel, resolution in PCS is extended to allow interchange of the members of a symmetrical pair in one of the clauses entering into the resolution.

To illustrate this extended definition of resolution, a simple theorem of elementary group theory will be proved.

> In any group (for which left identity and left inverses are postulated) the left cancellation law holds.

The premises are:

A. $\forall x \forall y \forall z (m(m(x,y),z) = m(x,m(y,z))$

ID. $\forall x (m(e,x) = x)$

IN. $\forall x (m(i(x),x) = e)$

and the conclusion is:

T. $\forall x \forall y \forall z (((m(x,y) = m(x,z)) \rightarrow (y = z))$

In PCS clausal form with the conclusion denied:

1. $\neg m(x,y,u) \vee \neg m(u,z,v) \vee \neg m(y,z,w) \vee m(x,w,v)$

2. $\neg e(x) \vee m(x,y,y)$

3. $\neg e(x) \vee \neg i(y,z) \vee m(z,y,x)$

4. $\neg a(x) \vee \neg b(y) \vee \neg c(z) \vee \neg m(x,y,u) \vee m(x,z,u)$

5. $\neg b(x) \vee \neg c(x)$

Note that a, b, c, e, i, m are all singular predicates.

The following sequence of clauses, appended to the premises, is a refutation. The justification for each clause is given in parentheses. The literal involved in resolution is underlined. When interchange of a symmetrical pair is involved, for clarity it is shown on a separate line.

6. $\neg e(u) \vee \neg i(y,x) \vee \underline{\neg m(u,z,v)} \vee \neg m(y,z,w) \vee m(x,w,v)$ (resolve 1,3)

7. $\neg i(y,x) \vee \underline{\neg m(y,z,w)} \vee m(x,w,z)$ (resolve 2,6)

8. $\neg a(x) \vee \neg b(y) \vee \neg c(z) \vee \underline{m(x,y,u)} \vee \neg m(x,z,u)$ (interchange symmetrical pair in 4)

9. $\neg a(y) \vee \neg b(z) \vee \neg c(v) \vee \neg i(y,x) \vee \neg m(y,v,w) \vee \underline{m(x,w,z)}$ (resolve 7,8)

10. $\neg m(x,y,u) \vee m(u,z,v) \vee \neg m(y,z,w) \vee \underline{\neg m(x,w,v)}$ (interchange symmetrical pair in 1)

11. $\neg a(y) \vee \neg b(z) \vee \neg c(v) \vee \neg i(y,x) \vee \underline{\neg m(x,y,u)} \vee m(u,v,z)$ (resolve 9,10)

12. $\neg e(u) \vee \neg b(z) \vee \neg c(v) \vee m(u,v,z)$ (resolve 3,11)

13. $\neg e(u) \vee b(z) \vee \neg c(v) \vee \underline{\neg m(u,v,z)}$ (interchange symmetrical pair in 12)

14. $\underline{b(v)} \vee \neg c(v)$ (resolve 2,13)

15. $\Box$ (resolve 5,14)

9 **Conclusion** PCS appears to offer certain computational advantages relative to conventional predicate calculus for automated reasoning. Specifically, unification is supplanted by simple alphabetic conversion; the need for an occur check disappears; 'cyclic terms' are properly represented; the binding environments required for the refutation process are less complex; and reasoning with identicals is simplified in that substitution of identicals is subsumed by resolution. The reason for this appears to be that the singular expressions involved in resolution-based reasoning in PCS function as generalized and "flattened" terms.

However, a complexity analysis for reasoning in PCS was not presented. Nor was the use of heuristics to guide the refutation process considered. Some heuristics developed for conventional predicate calculus are applicable. Others must be adapted to reasoning in PCS. Still others are peculiar to PCS. These topics are deferred to subsequent papers.

References

[1] Andrews, Peter B. *An Introduction to Mathematical Logic and Type Theory*, Academic Press, Orlando 1986.

[2] Barnes, Donald W. and John M. Mack *An Algebraic Introduction to Mathematical Logic*, Springer-Verlag, New York 1975.

[3] Chang, Chin-Liang and Richard Char-Tung Lee *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York 1973.

[4] Colmerauer, A. "Prolog and Infinite Trees," in Clark and Tarnlund (eds.) *Logic Programming*, Academic Press, New York 1982.

[5] Enderton, Herbert B. *A Mathematical Introduction to Logic*, Academic Press, New York 1972.

[6] Purdy, William C. *A Logic of Singular Predicates*, Report SU-CIS-91-32. School of Computer and Information Science, Syracuse University, 1991.

[7] Purdy, William C. *On the Question 'Do We Need Identity?'*, Report SU-CIS-91-33. School of Computer and Information Science, Syracuse University, 1991.

[8] Robinson, J. A. "A Machine-Oriented Logic Based on the Resolution Principle," *JACM* 12, 1965.