A property-list representation for certain formulas in predicate calculus

by

Erik J Sandewall

UPPSALA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCES

UPPSALA UNIVERSITY
COMPUTER SCIENCES DEPARTMENT
REPORT NR 18
JANUARY, 1969

A property-list representation for certain formulas in predicate calculus

by

Erik J Sandewall

(for an abstract, see next page)

Documentation page.

Title A property-list representation for certain formulas in predicate calculus

Author name Erik J Sandewall

Research performed at: Uppsala University, Uppsala, Sweden*

Abstract This paper describes

(1) a logical language for use in property-list-type data bases in question-answering systems. The language can handle binary relations, universal and existential quantifiers, " ε -quantifiers", and some implications.

(2) inference rules for this language.

(3) a proof procedure specially designed for property-list type representations. The procedure is complete at least with respect to a certain subset of the inference rules. It is a rather simple AND/OR tree search, so that previous work in heuristics is immediately applicable to it.

Key words and phrases: AND/OR tree, data base, decision procedure, proof procedure, property-list, property-set, question-answering.

CR categories: 3.64, 3.66, 5.21

* **Mail address:** Sturegatan 43 2 tr
752 23 Uppsala, Sweden

Sponsors: This research was supported in part by the Swedish Natural Science Research Council (contract Dnr 2711-6) and by the (Swedish) Research Institute of National Defense (beställn. 715411, 719925).

Introduction.

Early question-answering systems often used ad hoc representations for their data bases, and corresponding ad hoc inference methods for the question answering. For example, the SIR system ({Raphael 1964a}) represents binary relations on property-lists (this term will be defined below). In recent years, it has been argued (e.g. in {Slagle 1965b} and {Green 1968a}) that a dialect of predicate calculus should be used instead. This would have two advantages: (1) predicate calculus is a richer language, i.e. more things can be said in it; (2) for predicate calculus, one knows reasonably efficient proof procedures, e.g. resolution (for an introduction to resolution, see {Robinson 1965a}).

The distinction between these two approaches is of course not perfectly clear-cut. Even if one uses predicate calculus notation, he may find it useful as the data base grows to construct, for each object symbol c, a chained list of all literals or clauses where c occurs. This chained list is then a property-list for c. However, it remains that predicate calculus is not a particularly computer-oriented language in itself. One should therefore continue to give at least some attention to the possible use of other representations.

The purpose of the present paper is to demonstrate how property-list notation can be used in a more systematic manner than before. We shall consider both the epistemological problem ("how much can be said in a property-list-type notation?) and the inference problem ("how should the computer prove facts and answer questions from information expressed in property-list notation?"). An ultimate goal is that property-list notation shall no longer be considered as an ad hoc notation.

Corrections to "A property-list representation for certail formulas in predicate calculus"

page	line	says	change to
15	8	I7	I8
	9	I8	I9
	10	I9	I10
	then insert between lines 7 and 8 the following line: (I7) $\overleftrightarrow{R} \sqsubset \rightarrow \overleftrightarrow{R}$		
	-10	I9	I10
16	10	I9	I10
	-5	48	56
	-2	... , or (I9) ...	... , (I9), or (I10) ...

1. Conventional property-list language

Let $\bar{U} = \{\bar{u}, \bar{v}, \bar{w}, ,,,\}$ be a set of objects, and let $\bar{\Delta} = \{\bar{P}, \bar{R}, \ldots\}$ be a set of binary relations on $\bar{U}$, i.e. subsets of $\bar{U}\times\bar{U}$. Let $U = \{u,v,w \ldots\}$ be a set of distinguishable symbols, and let there be a mapping which assigns a member of $\bar{U}$ to each symbol. We shall understand that $\bar{u}$ is assigned to u, etc. Define Δ, P, R, ... in a corresponding manner(x).

Each member of $U\times\Delta\times U$ is called a <u>sentence</u>. A sentence uRv is called a <u>fact</u> iff $\langle\bar{u},\bar{v}\rangle \in \bar{R}$. Consider now the problem of representing a set of facts in computer memory on such a form that they can easily be retrieved (e.g. in order to compute the answer to a question).

This problem has been encountered by various workers in the question-answering field. When Lindsay saw it ({Lindsay 1963a}), $\bar{U}$ was a set of people and families, and the relations were "u is the husband in the family v", "u is an offspring in the family v", etc. Raphael encountered the same problem ({Raphael 1964a}), with $\bar{U}$ being a set of objects and people, and the relations being e,g. "u is physically part of w", "v is the owner of x", etc. Levien saw it ({Levien 1965a}) with $\bar{U}$ a set of people, meetings, institutions, and documents, and relations such as "u is the author of v","u is employed by w", etc. We met the same problem ourselves when we decided to translate natural-language sentences like "A gave B to C" into an expression

$$(\exists x)\ R_1(x,A) \wedge R_2(x,Give) \wedge R_3(x,B) \wedge R_4(x,C)$$

(x) The bar will sometimes be omitted, if no confusion can arise.

where x is the activity described ty the sentence, R_1 can crudely b e described as an "activity-to-its-subject" relation, R_2 is the "activity-to-its-verb" relation, etc.

One standard way of representing binary relations in the computer is through <u>property structures</u>. We formally define a property structure as a mapping

$$\sigma : \quad U \rightarrow \quad 2^{\Delta \times U}$$

i.e. a mapping which assigns a set of pairs Rv to each member u of U. A property structure σ <u>corresponds to</u> a set ϕ of facts iff

$$uRv \in \phi \quad \equiv \quad Rv \in \sigma(u)$$

If σ is a property structure and $Rv \in \sigma(u)$, we shall say that u <u>has the property</u> Rv in σ .

To represent a property structure in memory, one usually does as follows: a unique cell is associated with each member of U. A cell which is so associated is called an <u>atom</u>. The atom associated with u will itself be called u. A property Rv is represented as an indicator for R plus the address of v. All the properties that an atom u has are stored in such a way that they can be accessed from u as easily as possible. This may be done using a sequential list (in which case each σ(u) is represented as a <u>property-list</u>), through hatch-coding, or by other means.

If property structures are used, one should see to it that the set of relations is closed under reversion, i.e. that for each $R \in \Delta$ there exists some Я $\in \Delta$ such that uRv is a fact iff vЯu is. Also, it is desirable that the set ϕ of known facts is closed under reversion in a similar manner. The property structure σ corresponding to ϕ will then satisfy

$$Rv \in \sigma(u) \quad \equiv \quad Яu \in \sigma(v)$$

If R is a symmetric relation, then R and Я are the same relation. We shall use symmetric symbols (V,□,...) for symmetric relations.

Figure 1 illustrates how ϕ = {uPv, vꟼu, vRw, wЯv} can be represented as a property-list structure. Arrows stand for address references.

This terminates our description of conventional property-set and property-list representation. The reader will notice that nothing has been said about the problem of *inference* from property-set represented information. This mirrors the fact that, although several authors have utilized property-set representation for their programs, there does not (to our knowledge) exist any work on the *general* problem of inference from property-set represented information. But as we shall see in this report, some general techniques can be given.

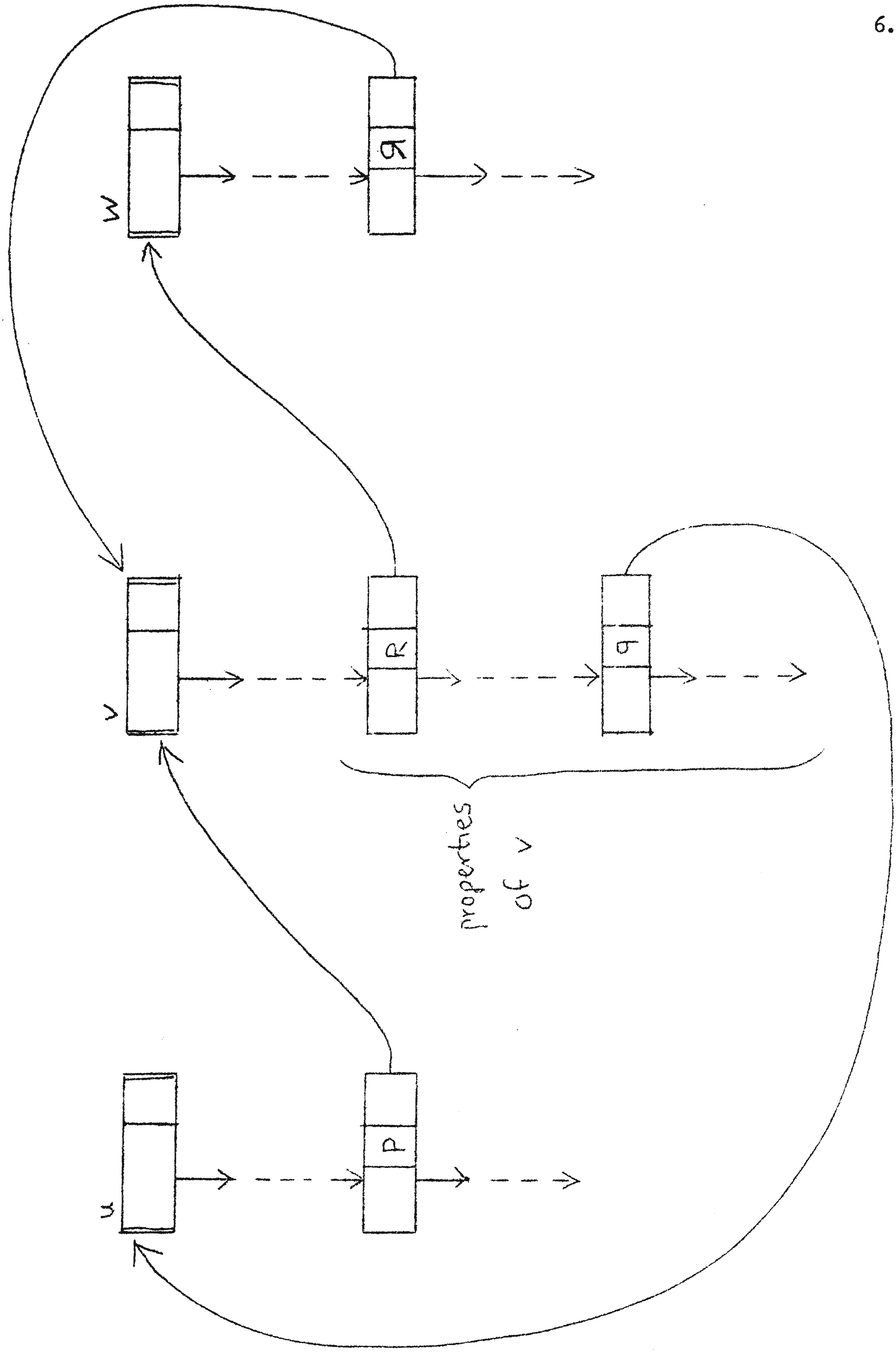


Figure 1.

2. Notation and approach

In this section, we shall first introduce some notation that will be needed for succeeding sections, and then give a summary of those sections, using the notation.

Throughout the report, we shall concentrate on inference rules on the form

$$xRy, yPz \vdash xQz$$

where x,y, and z are variables for members of U, and R, P, and Q are constant, not necessarily distinct relations. Inference rules that take this form will be called chaining rules. For chaining rules, we shall use the more compact notation,

$$R\ P \rightarrow Q$$

In terms of properties, this chaining rule says that if u has property Rv and v has property Pw, then u can be assigned property Qw. Q is called a product of R and P. A pair R P of relations may have no, one, or several products.

Our preference for chaining rules will become apparent both in the epistemological parts (we shall prefer relations whose properties can be characterized by such rules) and the inferential parts (we shall give proof methods which assume all inference rules to be on this form, and which have to be "patched" for each inference rule that takes on another form).

Let a set Γ of chaining rules be given. If $R\ P \rightarrow Q$ is in Γ, we write

$$Q_1\ Q_2 \ldots Q_{j-1}\ R\ P\ Q_{j+1} \ldots Q_k \Rightarrow Q_1 \ldots Q_{j-1}\ Q\ Q_{j+1} \ldots Q_k$$

Moreover, we write $\Pi \overset{x}{\Longrightarrow} \Sigma$ iff Π and Σ are sequences of relations, and either of the following holds:

(a) $\Pi = \Sigma$

(b) $\Pi \Rightarrow \Sigma$

(c) there exists some T such that $\Pi \Rightarrow T$ and $T \overset{x}{\Longrightarrow} \Sigma$.

Let $P_i v_i \in \sigma(v_{i-1})$ for $i = 1,2,\ldots k$. We then say that v_0 has the <u>implicit property</u> $P_1 P_2 \ldots P_k v_k$ in σ.

This is all notation we need for the moment. We shall now use it to give a short and rather abstract summary of what will be done in the next few sections. The summary will use the concepts and the notation of {Ginsburg 1966a}. However, the following sections will not be based on Ginsburg nor on this summary. Readers who so desire can therefore safely skip from here to the beginning of next section.

Let U, Δ, ϕ, and Γ be given as before; let vQw be an arbitrary sentence, and consider the decision problem

$$\text{Does} \quad \phi \vdash_{\Gamma} \quad vQw \ ?$$

If σ is the property structure associated with ϕ , this problem can be phrased: Does v have any (possibly implicit) property Πw in σ such that $\Pi \overset{x}{\Longrightarrow} Q$? Let L be the set of all Π such that $\Pi \overset{x}{\Longrightarrow} Q$. It is easily seen that L is a context-free language, generated by the following grammar:

terminal symbols: members of Δ

non-terminal symbols: $\underline{R}$, for each R in Δ

productions: $\underline{S} \rightarrow \underline{R}\ \underline{P}$, for each $R\ P \rightarrow S$ in Γ

$\underline{R} \rightarrow R$, for each R in Δ

initial symbol: $\underline{Q}$

The given decision problem, i.e. "Does v have any (possibly implicit) property Πw such that Π is in the language L?" is clearly a parsing problem. Since the implicit properties Πx of v can be scanned from left to right in the property structure, one can use conventional parsing schemes (which are essentially push-down acceptors) for solving the decision problem.

Consider now the right-linear language (= regular set) L' which is generated by the grammar of L, except that productions

$\underline{S} \rightarrow \underline{R}\ \underline{P}$

have been changed into

$\underline{S} \rightarrow R\ \underline{P}.$

Clearly, $L' \subseteq L$. If $L' = L$, i.e. if L is a regular set, then the decision problem can be solved using a finite-state acceptor. This speeds up the parsing process considerably. In a vocabulary familiar to LISPers, we have eliminated a case of "double recursion".

In sections 3-6 of this report, we shall do two things:

(1) Give a class of relations and associated inference rules for which we do have $L' = L$;

(2) Work out the details of the finite-state acceptor that will answer questions in these relations.

In order to reach a wider class of readers, we shall not use acceptors and formal languages in our description, but turn to a more direct notation.

Sections 7 and 9 will be devoted to extending the property structure "language".

3. How to handle common subsets of properties

Let u and w be two objects which have a considerable number of properties in common, i.e. we have

uRv, uPy, zQu, ...

as well as

wRv, wPy, zQw, ...

To avoid dublication of the common subset of properties (Rv, Py, ρz, ...), we would like to break it out as a common sublist of the property-lists of u and w (or to have a similar device if other than property-list representations of the property structure are used). Of course, we also want to do this when more than two objects have common properties.

A correct way of obtaining such sublists would be the following:

(1) Permit atoms for subsets of $\bar{U}$, not merely for members of $\bar{U}$;

(2) Introduce ε (set membership) as one more binary relation;

(3) For each binary relation R, introduce a new relation R^+ defined through

$$aR^+v \quad =_{def} \quad (\forall\ x \in a)\ xRv$$

In particular, ε^+ is the subset relation.

The common sublist of u and w can then be obtained by introducing a new atom m for which

$$\sigma(m) = \{R^+v,\ P^+y,\ \rho^+z,\ \ni u,\ \ni w\}$$

so that $u \in m$, $w \in m$. All the common properties that u has can be substituted by the property εm, and similarly for w. We need inference rules like

$$\varepsilon \; R^{+} \rightarrow R$$

and

$$\varepsilon^{+} \; R^{+} \rightarrow R^{+}$$

Naturally, such inference rules would be used in an implicit manner when a question is being answered, rather than explicitly by adding more properties to the property-lists.

If this approach were to be used in a systematic way, we would need, besides R^{+}, relations for

$$(\forall y \varepsilon b) \;\; vRy$$

and for

$$(\forall x \varepsilon a)(\forall y \varepsilon b) \;\; xRy$$

The number of relations and corresponding inference rules would be unnecessarily large. We shall therefore adopt a modified approach, which will also be defined slightly more strictly than the above.

Let $\bar{U}$ and $\bar{\Delta}$ be given like at the beginning of section 1. Let V be a set of symbols, and let there be a mapping which assigns a <u>subset</u> $\bar{a}$ of $\bar{U}$ to each symbol a. Let Δ be a set of symbols which consists $\subset$, $\supset$, and (for each symbol $\bar{R}$ in $\bar{\Delta}$) R, $\tilde{R}$, Я, and Я̃ . Every member of $V \times \Delta \times V$ is called a <u>sentence</u>. A sentence is called a <u>fact</u> iff it satisfies some of the following conditions:

(a) The sentence $a \subset b$ is a fact iff $\bar{a}$ is a subset of $\bar{b}$, and similarly for $b \subset a$;

(b) The sentence aRb is a fact iff

$$(\forall x \varepsilon a)(\forall y \varepsilon b) \;\; xRy$$

(c) The sentence $a\tilde{R}b$ is a fact iff

$$(\forall x \varepsilon a)(\forall y \varepsilon b) \sim xRy$$

(d) $\reflectbox{R}$ denotes the reverse relation of R, and similarly for $\tilde{\reflectbox{R}}$.

The members of Δ except $\subset$ and $\supset$ will be called <u>regular relations</u>.

For each fact in our old sense of the word, there exists a corresponding fact in the new sense. Let $\overset{\circ}{u}$ in V denote the set whose only member $\bar{u}$ is. Then $\overset{\circ}{u}R\overset{\circ}{v}$ is a fact iff uRv is.

If R is regular and $\bar{a}$ or $\bar{b}$ is an empty set, then both aRb and $a\tilde{R}b$ are facts. Moreover, if aRb and cRb are facts, and $\bar{a}$, $\bar{b}$, and $\bar{c}$ are non-empty sets, then neither $(a \cup c)Rb$ nor $(a \cup c)\tilde{R}b$ is a fact. The $\sim$ superscript is not ordinary negation, therefore. The semantics of this logic can be worked out correctly using four truth-values, {t}, {f}, {t,f}, and ϕ (the empty set). If $\tau(A)$ stands for "the truth-value of A", we have

$$\tau(\overset{\circ}{u}R\overset{\circ}{v}) = \{\tau(uRv)\}$$

$$\tau((a \cup c)Rb) = \tau(aRb) \cup \tau(cRb)$$

A formula is then said to be a fact iff its truthvalue is {t} or ϕ.
- Such systematic treatment of the semantics lies beyond the scope of the present report. Let us remark, however, that the same four-valued logic has been used as the basis of the author's LISP A, an incremental computer languege (see {Sandewall 1968c}) . Notice also that the logic here vaguely resembles the logic of Quine's ι operator.

In what follows, all relations are therefore restricted to taking sets as arguments.

We immediately obta in the following <u>inference rules</u>:

(I1) $\subset \subset \;\rightarrow\; \subset$

(I1') $\supset \supset \;\rightarrow\; \supset$

(I2) $\subset R \;\rightarrow\; R$

(I2') $R \supset \;\rightarrow\; R$

In (I2) and (I2'), R is an arbitrary regular relation. The prim'ed rules can be dispensed with if we notice the following meta-rule:

<u>Rule of reversion.</u> If P, Q, and R are arbitrary relations, and if

$$R\ P \;\rightarrow\; Q$$

then

ꟼ Я → ϙ

4. Notation for existence.

In the preceeding section, we introduced a property-oriented notation that took care of some cases where predicate calculus would use universal quantifiers. In the present section, we shall introduce notation (1) for saying "this set is (is not) empty" and (2) for handling some cases where predicate calculus would use existential quantifiers.

The relation $\Box$ is defined as follows: $a \Box b$ is true iff $a \cap b$ is the empty set, and false otherwise. In particular, $a \Box a$ is true iff a is the empty set. The relation $\tilde{\Box}$ is defined by
$a \tilde{\Box} b =_{def} \sim (a \Box b)$

It immediately follows that both $\Box$ and $\tilde{\Box}$ are symmetric, and that we have the inference rules

(I3) $\subset \Box \rightarrow \Box$

(I4) $\tilde{\Box} \subset \rightarrow \tilde{\Box}$

as well as

(J1) $a \tilde{\Box} b \vdash b \tilde{\Box} b$

(J2) $a \Box a \vdash aRb$

(J3) $a \Box a \vdash a \Box b$

(J4) $a \Box a \vdash a \subset b$

Rules (J1) to ( ) fail to fit into the desired pattern for inference rules and will largely be ignored. Let us now proceed to the counterpart of the existential quantifier. Let R be a regular relation. We define the relations $\vec{R}$, $\overleftarrow{R}$, and $\overleftrightarrow{R}$ as follows:

$$a\vec{R}b \quad =_{def} \quad (\forall x \in a)(\exists y \in b)\ \mathring{x}R\mathring{y}$$

$$a\overleftarrow{R}b \quad =_{def} \quad (\forall y \in b)(\exists x \in a)\ \mathring{x}R\mathring{y}$$

$$a\overleftrightarrow{R}b \quad =_{def} \quad (\exists x \in a)(\exists y \in b)\ \mathring{x}R\mathring{y}$$

and obtain the inference rules:

(I5) $\quad \subset \vec{R} \rightarrow \vec{R}$

(I6) $\quad \vec{R} \subset \rightarrow \vec{R}$

(I7) $\quad R\,\tilde{\Box} \rightarrow \vec{R}$

(I8) $\quad \tilde{\Box}\,\vec{R} \rightarrow \overleftrightarrow{R}$

(I9) $\quad \vec{R}\,\tilde{Я} \rightarrow \Box$

(J5) $\quad a \Box a \quad \vdash \quad a\vec{R}b$

(J6) $\quad a\overleftrightarrow{R}b \quad \vdash \quad b\tilde{\Box}b$

(J7) $\quad a\vec{R}b,\ b \Box b \quad \vdash \quad a \Box a$

When using the rule of reversion, we notice that $\overleftarrow{Я}$ is the reverse of $\vec{R}$, and $\overleftrightarrow{Я}$ is the reverse of $\overleftrightarrow{R}$. When using (I9) for $R = \tilde{P}$, we naturally take $\tilde{R}$ to mean P.

With these conventions, we manage to express existence within the framework of property-sets, and in such a way that the important inference rules are chaining rules.

<u>Remark 1.</u> For each relation Q, either of the rules

$$\subset Q \rightarrow Q$$

$$\supset Q \rightarrow Q$$

holds. Therefore, it is sufficient to express equality $a=b$ as $a \subset b$, $a \supset b$.

<u>Remark 2</u>. The relations aRb, $a\vec{R}b$, $a\overset{\rightarrow}{\tilde{R}}b$, $a\tilde{R}b$ can be characterized as "each member of the set a is the relation R to all/some/ not all/no members of the set b". It would be natural to extend the notation to other quantities, like "exactly one" (which is sometimes written in predicate calculus as $\exists_1$) or "exactly five" (compare Raphael's representation of "every hand has exactly five fingers as part"). If such specific quantities are accepted, remark 1 no longer holds.

<u>Remark 3</u>. For the fun of it, we can write $\subset$ as $\overset{\rightarrow}{\tilde{\square}}$. Rules (I5) and (I6) the both specialize as (I1); (I9) specializes as (I3), and (J5) specializes as (J4).

After this new notation has been introduced, we must update some of the old definitions. Let $\bar{\Delta}$ be a set of relations like in previous sections, and let it have L members. The set ∇ and Γ are defined as follows:

∇ is the set of $4 + 16$ L relation symbols obtained as follows:

(a) $\subset$, $\supset$, $\square$, and $\tilde{\square}$ are members of ∇;

(b) If R is a regular relation in Δ, then R, $\vec{R}$, $\overleftarrow{R}$, and $\overleftrightarrow{R}$ are members of ∇.

Γ is the set of $6 + 48$ L inference rules obtained as follows:

(a) (I1), (I3), and (I4) are members of Γ;

(b) If K is a regular relation in Δ, then each inference rule obtained from (I2), (I5), (I6), (I7), (I8), or (I9) by substituting K for R, is a member of Γ;

(c) If γ is an inference rule in Γ and γ' is obtained from γ by the rule of reversion, then γ' is in Γ .

Any member of $V\times\nabla\times V$ will be called a sentence. A sentence is called a fact iff it satisfies the definitions on page 11/12 viz 14/15. A property structure is a mapping

$$\sigma : \quad V \rightarrow 2^{\nabla\times V}$$

Other definitions remain unchanged.

If ϕ is a set of sentences and aQb is a sentence, $\phi \vdash aQb$ will mean "aQb can be inferred from ϕ using the inference rules in Γ ". (Notice that the rules (J1) to (J7) are ignored).

Let us now proceed to the problem of using the inference rules Γ on a property structure σ .

5. Verification of properties.

Let V, ∇, Γ, and ϕ be given as before. If aQb is an arbitrary sentence, then the expression

aQb ?

will be called a question. The answer to the question is either of the symbols Yes, No, or Nil ($\sim$ I do not know), and is defined as follows:

$$\begin{cases} \text{If } \phi \vdash aQb & \text{the answer is Yes} \\ \text{If } \phi \vdash \sim aQb & \text{the answer is No} \\ \text{Otherwise the answer is Nil} \end{cases}$$

If Q is $\subset$, $\supset$, $\vec{R}$, or $\overleftarrow{R}$, then $\sim$ aQb can not be represented as a single relation in the language of the last two sections. We shall therefore need two question-answering procedures: one verification procedure which determines whether the answer may be Yes, and one rejection procedure which determines whether the answer may be No. This section will be concerned with the verification procedure.

Remark. Our use of Nil for "Don't know" is motivated by LISP conventions. Let γ ? be a question, let verif(γ) have Yes or Nil as value, and let rejec(γ) have No or Nil as value. If V is the generalized LISP 'OR', the answer to γ ? is

verif(γ) V rejec(γ)

Let σ be the property structure corresponding to ϕ. The following would seem to be a reasonable verification procedure for the question aQb ? : First check whether $Qb \in \sigma(a)$, and if so, answer Yes. Otherwise, for each inference rule $P\ S \rightarrow Q$, work through all properties Pc that a has or can be inferred to have, and ask whether the referred-to c has or can be inferred to have the property Sb.

This method will soon explode, mainly due to the "double recursion", represented by the double occurrence of "or can be inferred to have" in the description. For an extreme example, suppose Q is transitive, so that $Q\ Q \rightarrow Q$. Also, suppose aQa_1, a_1Qa_2, ... $a_{k-1}Qa_k$ are stored in σ , but not a_jQb for any j. The above method will run through 2^k branches in the search tree before it gives up trying to prove aQb.

However, it is easily verified (e.g. by having a computer program run through all possible choices of P, Q, and R) that the set Γ satisfies the following

Associativity condition: Each product $((P\ Q)\ R)$ is also a product $(P\ (Q\ R))$, and vice versa.

Because of this, we can remove the first (but not both the first and the second) occurrence of "or can be inferred to have" in the above method without weakening it at all. We shall prove this result by specifying a verification method which utilizes the associativity condition, and which clearly does not perform double recursion. - The procedure is specified as follows:

A. The question aQb ? is expressed by assigning the verification property $\check{Q}a$ to b.

B. We use the inference schema

(Q1) $\overset{\times}{\rho}Q \to I$

(where Q is an arbitrary member of ∇), and all rules obtained by the meta-rule

<u>Rule of verification:</u> If $P\,S \to Q$, then $\overset{\times}{\rho}\,P \to \overset{\times}{Q}$

C. To verify the question, use the verification property and the inference rules to generate more properties for b. If it obtains the property Ib, then the answer is Yes(*).

It is clear that this method does not perform double recursion. In the above example with transitive Q, it will assign the property $\overset{\times}{\rho}b$ to a, a_1, ... a_k, i.e. search k+1 branches in the tree instead of 2^k. Let us now see how it works for the question aQb ? in a couple of cases.

1. Qb ε σ(a). Inference rule (Q1) immediately gives the answer Yes.

2. a has property Pc, c has property Sb, and $P\,S \to Q$ is in Γ. By the rule of verification, we have $\overset{\times}{\rho}\,P \to \overset{\times}{Q}$, so b successively obtains the properties $\overset{\times}{\rho}a$, $\overset{\times}{Q}c$, Ib.

3. a has property $\subset$c, c has Qd, d has $\supset$b, and Q is a regular relation. B successively obtains the properties $\overset{\times}{\rho}a$, $\overset{\times}{\rho}c$, Id (which is useless), $\overset{\times}{\subset}$ d, Ib (which yields the answer Yes). Notice that (I2) and the rule of verification give $\overset{\times}{\rho}Q \to \overset{\times}{\subset}$, so $\overset{\times}{\rho}Q$ has two products.

(*) In a computer implementation of the verification procedure, it is possible but not necessary to store verification properties among other properties. It will usually be more efficient to represent them as subroutine calls, i.e. on a push-down list.

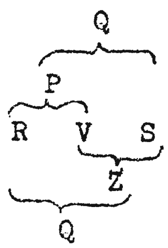
4. a has property Rd, d has Ve, e has Sb, and P S $\rightarrow$ Q, R V $\rightarrow$ P are in Γ . By the rule of verification, $\check{Q}$ P $\rightarrow$ $\check{S}$ and $\check{P}$ R $\rightarrow$ $\check{V}$. However, since we do not have any rule on the form $\check{Q}$ R $\rightarrow$ ... , it seems that we have got stuck.

This is where the associativity condition comes in. It guarantees that there exists some Z in ∇ such that the following are in Γ :

V S $\rightarrow$ Z

R Z $\rightarrow$ Q

These rules can be summarized as follows:



The rules derived by the rule of verification give to b the successive properties $\check{Q}$a, $\check{Z}$d, $\check{S}$e, Ib,

It is trivial that if the associativity condition holds for all sequences of three relations, then it also holds for any longer sequence. Therefore, the method given in steps (A) to (C) is complete with respect to the rules in Γ , i.e. it will answer Yes to the question γ iff $\phi \vdash \gamma$.

It should be noticed that the verification method here works for any set of rules that satisfy the associativity condition. It is therefore a proof method for property-set represented information in general.

6. Rejection of properties.

A sequence Π of relation symbols is said to be <u>contradictory</u> iff it is a contradiction that a set b should have the implicit property Πb. For the set ∇ of relation symbols, there are no contradictory sequences of length 1, but the following four types of sequences of length 2:

$\Box\tilde{\Box}$ $\quad\quad$ $\tilde{\Box}\Box$

$\tilde{R}\,\overleftrightarrow{\text{Я}}$ $\quad\quad$ $\overleftrightarrow{R}\,\tilde{\text{Я}}$

where R is an arbitrary, regular relation. Moreover, if $\Sigma \overset{x}{\Longrightarrow} \Pi$ and Π is contradictory, so is Σ . We believe in the following

<u>Hypothesis</u>: Every contradictory sequence Σ in ∇ satisfies $\Sigma \overset{x}{\Longrightarrow} \Pi$, where Π is one of the four contradictory sequence schemas given above.

The following are examples of contradictory sequences which agree with the hypothesis:

$\text{Я}\,\tilde{\Box}\,\tilde{R}\,\tilde{\Box}$

$\vec{\text{Я}}\,\tilde{R}\,\tilde{\Box}$

$\text{Я}\supset\supset\tilde{\Box}\,\overleftrightarrow{R}$

$\tilde{\Box}\;\vec{R}\,\tilde{\text{Я}}$

The following method will answer No to the question aQb ? in some of the cases where $\phi \vdash {}^{\sim}$ aQb. If the hypothesis is correct, it will answer No in all cases where

ϕ , aQb $\vdash$ (a has an implicit property Σa)

where Σ is a contradictory sequence.

A. The question aQb ? is expressed by assigning the property $\overset{\infty}{\rho}a$ to b.

B. In general, $\overset{\infty}{\rho}a \in \sigma(b)$ is taken to mean "assume that b has the property ρa". $\overset{\infty}{\rho}a$ is called an <u>assumed property</u>.

C. The statement "cRb is sufficient for rejecting the original question" is expressed by assigning the <u>rejection property</u> $\overset{=}{\mathcal{R}}c$ to b.

D. We use the inference schema

$$\overset{=}{\rho}\, Q \;\rightarrow\; \tilde{I}$$

and all inference rules obtained by the meta-rules

<u>Rule of assumption</u>: If $P\ S \rightarrow Q$, then $\overset{\infty}{P}\ S \rightarrow \overset{\infty}{Q}$.

<u>Rule of redoubt</u>: If $P\ S$ is a contradictory sequence, then $\overset{\infty}{P} \rightarrow \overset{=}{\mathcal{Z}}$.

<u>Rule of rejection</u>: If $P\ S \rightarrow Q$, then $\overset{=}{\rho}\, P \rightarrow \overset{=}{\mathcal{Z}}$.

E. To reject the question aQb ?, use the originally assigned property $\overset{\infty}{\rho}a$ and the inference rules to generate more properties for b. If b obtains the property $\tilde{I}b$, then the answer is No.

The idea behind the method is as follows: suppose a has some implicit property Πb such that $\rho\Pi$ is contradictory. By our hypothesis, there exist sequences Π_1 and Π_2 such that $\Pi_1\Pi_2 = \rho\Pi$ and such that Π_1 has a product P_1 and Π_2 has a product P_2, where P_1P_2 is a contradiction. Using associativity, $\rho\Pi$ can therefore be written

$$((\ldots((P_{11}\ P_{12})\ P_{13})\ \ldots\ )P_{1k})(P_{21}(P_{22}\ \ldots\ (P_{2,j-1}\ P_{2j})\ldots))$$

where of course the P_{1i} constitute Π_1 and the P_{2i} constitute Π_2.

The method given in steps A to E eats the first sub-sequence using assumed properties and the rule of assumption; uses the rule of redoubt to switch to rejection properties; and then works through the second sub-sequence using the rule of rejection.

<u>Remark</u>: One might be tempted to eliminate the use of rejection properties by writing simply

$$\overset{\infty}{P}\, S \rightarrow \tilde{I}$$

for each contradictory sequence P S. Unfortunately, this does not work since the resulting, extended set of inference rules is not associative. For example, such a method would not recognize the contradictory sequence

$$\overset{\underset{\sim}{\infty}}{R}\, \tilde{\square}\, \vec{Я}$$

because the parsing $((\overset{\underset{\sim}{\infty}}{R}\, \tilde{\square})\, \vec{Я})$ gives $\overset{\infty}{\overset{\rightleftarrows}{R}}\, \vec{Я}$ which is not a contradiction. Using rejection properties, we recognize this as a contradiction through the parsing $(\overset{\underset{\sim}{\infty}}{R}\ (\tilde{\square}\ \vec{Я}))$.

7. Defined symbols.

A relation R is said to be distributive in its first argument iff the following two requirements hold:

1. If $c \subset a$ and aRb are facts, then cRb is a fact;
2. If aRb and cRb are facts, then $(a \cup c)Rb$ shall be a fact.

Distributivity in the second argument is defined similarly. Of the relations in ∇ , $\subset$ and $\vec{R}$ are distributive in their first arguments, whereas $\Box$ and regular relations R are distributive in both arguments.

Let P, S, and Q be distributive in their first arguments, and consider the implication

$$(\forall x) \quad \mathring{x}Pd \ \wedge \ \mathring{x}Se \ \supset \ \mathring{x}Qf$$

In terms of properties, this implication can be expressed: "If a property set includes the properties Pd and Se, then Qf can be added to it". We shall now generalize the property structure so that such implications can be expressed in it.

For each implication of the above form, we introduce a new symbol ξ which stands for "the set of all x for which $\mathring{x}Pd$ and $\mathring{x}Se$" (in lattice terms, ξ is the l.u.b. of all sets a such that $aPd \ \wedge \ aSe$). We clearly have

$$\sigma(\xi) = \{Pd, Se, Qf\}$$

Also, it is a rule that "if a property set includes the properties Pd and Se, then $\subset \xi$ can be added to it". Qf can then be added through chaining. It remains to provide a notation for the underlined rule.

We accomplish this by defining a mapping τ similar to σ, i.e. we have

$$\tau : \quad V \rightarrow 2^{V\times V}$$

τ shall be the mapping that assigns definitions to symbols like ξ. For conventional c in V which do not have any definition, $\tau(c)$ is the empty set. Therefore,

$$\tau(a) \subseteq \sigma(a) \qquad \text{for every } a \text{ in } V.$$

In the example given for introduction, we have

$$\sigma(\xi) = \{Pd, Se, Qf\}$$
$$\tau(\xi) = \{Pd, Se\}$$

Naturally, we also have

$$\reflectbox{P}\xi \; \varepsilon \; \sigma(d)$$

whereas $\tau(d)$ is the empty set; and similarly for e and for f. - If σ and τ have been constructed in this way from a set ϕ of sentences and inference rules, then the pair $\langle\sigma, \tau\rangle$ is called a (generalized) property structure. If $\tau(\xi)$ is not empty, ξ is called a <u>defined symbol</u>.

So much for notation. Let us now tackle the problem of inference using defined symbols. We start with the easiest case.

<u>Example</u>. ξ is defined like above, and c has properties Pd and Se. It is asked whether f has property $\reflectbox{Q}c$, i.e. c has been assigned the verification property $\overset{\times}{Q}f$. Conventional chaining gives c the property $\overset{\times}{\subset}\xi$. We immediately see that the following rule is sound:

(D1') If c has property $\overset{\times}{\subset}\xi$, and if $\tau(\xi)$ is not empty, then c can be assigned the property $\overset{\times}{P}d$ for each Pd in $\tau(\xi)$.

There is a complication. In previous sections, if a symbol c had several verification properties, these used to be "OR-connected", i.e. it was sufficient that some of them could be reduced to the property Ib or the answer Yes. But in rule (D1'), <u>all</u> the properties Pd in $\tau(\xi)$ must be verified. The verification procedure therefore gives and AND-OR tree, rather than a simple OR tree. Fortunately, such AND-OR trees have previously been studied, e.g. in {Slagle 1963a}, {Slagle 1968a}, and {Sandewall 1968d}. They do not present any difficulties in principle, but it does take some book-keeping to account for the obvious distributive etc. laws. Also, the heuristic problem of searching AND-OR search trees in an efficient way has been studied ({Slagle 1968a}). We shall not delve into such matters here, but merely assume that"the system" keeps track of AND-OR connections automatically.

Let us now modify the above example and assume that the same question had been formulated by giving f the property $\check{\wp}c$ instead. The only way to handle this situation seems to be through the following rather general rule:

(D2) If f has property $\check{\wp}c$ and ξ is a defined symbol, and if $\check{\wp}\subset \to \check{S}$, then we can assign two AND-connected properties: $\check{S}\xi$ to f, and $\check{\subset}\xi$ to c.

Because of its wide scope, this rule can of course not be applied indiscriminately. One must use heuristic criteria to determine for which $\check{\wp}c$ and which ξ it shall be used. Notice, in this context, that if it takes much effort to reduce $\check{\subset}\xi$ to a Yes (i.e. to prove that the implication that corresponds to ξ can be used), but only a little effort to prove that $\check{S}b$ <u>cannot</u> reduce to Yes (i.e. that the implication is useless in the situation), then any reasonably sophisticated heuristic system for handling the AND-OR connections would process $\overset{\times}{S}b$ rather soon and then abandon work on $\overset{\times}{\subset}\xi$.

Are rules (D1') and (D2) sufficient? Suppose b has a verification property $\overset{\times}{\wp}a$, and suppose a has an implicit property Πb, where $\Pi \overset{\times}{\Longrightarrow} Q$, which will yield an answer Yes to the question. When working with defined symbols, we are concerned about the following two cases, and need not be concerned about any other:

(1) $\Pi = (\Pi_1 \supset \Pi_2)$, where the $\supset$ should be derived by using a defined symbol. Rule (D1') clearly takes care of such cases exactly when Π_2 is the empty sequence.

(2) $\Pi = (\Pi_1 \subset \Pi_2)$, where the $\subset$ should be derived by using a defined symbol. Rule (D2) takes care of such cases for arbitrary Π_1 and Π_2 (except of course Π_1 and Π_2 which contain an $\supset$ where (D1') fails). (*)

Thus it remains to take care of case (1) for arbitrary Π_2. The following is a crude method:

(D1) If c has property $\overset{\times}{\wp}\xi$, if $\tau(\xi)$ is not empty, if g is an arbitrary symbol, and if
$\overset{\times}{\wp}\supset \;\rightarrow\; \overset{\times}{S}$, then we can assign two AND-connected verification properties: $\overset{\times}{S}g$ to c, and $\overset{\times}{\subset}\xi$ to g.

This method is of course even more generous than (D2), and we will be interested in methods to restrict the choice of g.

(*) To account for the case where Π_2 is empty, we must permit $\overset{\times}{S} = I$ in the specification of (D2).

8. WH questions.

Question-answering computer programs need to deal not only with YES/NO questions, but also with questions of the type "which ... have the properties ...? ". If the data base is a property structure σ , such questions can be answered by the following

Retrieval procedure. Let a set $T = \{P_1e_1, P_2e_2, \dots P_ke_k\}$ of properties be given, and let it be our task to retrieve symbols g in the data base, such that every g has, for $i = 1,2, \dots k$ an implicit/ property $\Pi_i e_i$ where $\Pi_i \overset{x}{\Longrightarrow} P_i$. Such g are retrieved by the following procedure:

A. Introduce a symbol ξ for which $\tau(\xi) = T$.

B. Assign to ξ the verification property $\check{P}_1e_1$, and let the verification procedure run.

C. To each g except ξ such that the property Ig is assigned to ξ in step B, assign the AND-connected verification properties $\check{P}_2e_2, \dots \check{P}_ke_k$. If all these verification properties lead to Ig, then g is an answer to the task.

Thus the verification procedure is useful for answering WH questions. Conversely, the above retrieval procedure is useful for heuristic purposes in the verification procedure, e.g. to restrict the choice of g in (D1). In order to make full use of the associativity of the AND's of (D1) and the retrieval procedure, we merge them into the following rule:

(D1 improved) If c has property $\check{\rho}\xi$, where ξ is a defined symbol, if $\sigma(\xi) = \{ P_1e_1, \dots P_ke_k\}$, and if $\check{\rho} \supset \rightarrow \check{S}$, then we can run the following procedure:

A. Step B of the retrieval procedure. The verification properties that occur are unrelated (neither AND- nor OR-related) to all other verification properties in the system.

D. To each g except ξ such that the property Ig is assigned to ξ in step A, assign the AND-connected properties $\overset{\times}{P}_2 e_2$, ... $\overset{\times}{P}_k e_k$, $\overset{\times}{\zeta} c$. This bundle of AND-connected properties is OR-connected with $\overset{\times}{\rho}\xi$.

9. A remark on systems of definitions.

The notation of previous section can handle some but not all expressions with two or more quantified variables. An example of an expression which it can handle, is(*)

$$(\forall x)(\forall y) \quad \mathring{x}Pd \;\wedge\; \mathring{y}Se \;\wedge\; \mathring{x}R\mathring{y} \quad \supset \quad \mathring{x}Qf$$

which can be re-written as

$$(\forall x) \left[(\exists y)\; \mathring{y}Se \;\wedge\; \mathring{x}R\mathring{y} \right] \quad \wedge \quad \mathring{x}Pd \;\supset\; \mathring{x}Qf$$

and then expressed through

$$\tau(\eta) \quad = \quad \{Se\}$$

$$\tau(\xi) \quad = \quad \{Pd,\ \vec{R}\eta\}$$

$$\sigma(\xi) \quad = \quad \{Qf,\ \ldots\ \}$$

An example of an expression which cannot be handled is

$$(\forall x)(\forall y) \quad \mathring{x}Pd \;\wedge\; \mathring{y}Se \;\wedge\; \mathring{x}R\mathring{y} \quad \supset \quad \mathring{x}Z\mathring{y}$$

We shall not here introduce any notation for such expressions in a systematic manner, but merely indicate the principles for such a notation.

Let a and b be two sets which satisfy

$$aPd \;\wedge\; bSe \;\wedge\; a\vec{R}b \;\wedge\; a\overleftarrow{R}b$$

It is easily verified that there exists a unique l.u.b. ξ for all such sets a, and similarly a unique l.u.b. η for all such sets b. These ξ and η can be used is defined sets, like in the previous section. Possibly, one could write

---

(*) We assume that P, S, R, and Q are distributive where necessary.

$$\tau(\xi) \quad = \quad \{Pd, \overrightarrow{R}b\}$$

$$\sigma(\xi) \quad = \quad \{Pd, \overrightarrow{R}b, \overleftarrow{R}b, \ldots\}$$

$$\tau(\eta) \quad = \quad \{Se, \overrightarrow{\text{Я}}a\}$$

$$\sigma(\eta) \quad = \quad \{Se, \overrightarrow{\text{Я}}a, \overleftarrow{\text{Я}}a, \ldots\}$$

However, some new devi e is needed to represent the relation $\mathring{x}Z\mathring{y}$ in a correct manner. It seems natural to introduce a third function besides σ and τ for this purpose. Accordingly, some further inference rules and some extensions to the verification and refutation procedures are needed.

10. A remark on non-chaining inference rules.

In sections 5 through 9, only chaining rules Γ have been used. We have taken the liberty to ignore rules (J1) to (J7), claiming that these can be accounted for by small modifications, "patches", to the verification and rejection procedures. In partial support of this claim, we shall give here the extra rules which are necessary in the verification procedure.

(J1) If some c has property $\overset{\times}{\rho}e$, e has property $\tilde{\Box}\,a$, and $\overset{\times}{\rho}\tilde{\Box} \to \overset{\times}{q}$, then c can be assigned the property $\overset{\times}{q}e$.

(J2 - J5, first rule) If c has property $\overset{\times}{\rho}a$, if $\overset{\times}{\rho}S \to \overset{\times}{q}$, and $a\Box a \vdash aSb$, then for each object symbol b, two AND-connected verification properties can be introduced:

$$\overset{\times}{\Box}a \ \varepsilon \ \sigma(a) \qquad \text{and} \qquad \overset{\times}{q}b \ \varepsilon \ \sigma(c)$$

(J2 - J5, second rule) If c has property $\overset{\times}{\rho}a$, if $\overset{\times}{\rho}S \to \overset{\times}{q}$, and $b\Box b \vdash aSb$, then for each object symbol b, two AND-connected verification properties can be introduced:

$$\overset{\times}{\Box}b \ \varepsilon \ \sigma(b) \qquad \text{and} \qquad \overset{\times}{q}b \ \varepsilon \ \sigma(c)$$

(J6, first rule) If c has property $\overset{\times}{\rho}e$, e has property $\overleftrightarrow{R}\,a$, and $\overset{\times}{\rho}\tilde{\Box} \to \overset{\times}{q}$, then c can be assigned the property $\overset{\times}{q}e$.

(J6, second rule) If c has property $\overset{\times}{\rho}e$, e has property $\overleftarrow{R}a$, and $\overset{\times}{\rho}\tilde{\Box} \to \overset{\times}{q}$, then two AND-connected properties can be introduced:

$$\overset{\times}{\Box}a \ \varepsilon \ \sigma(a) \qquad \text{and} \qquad \overset{\times}{q}e \ \varepsilon \ \sigma(c)$$

(J7) If c has property $\overset{\times}{\wp}a$, $\overset{\times}{\wp}\square \rightarrow \overset{\times}{q}$, and a has property $\vec{R}b$, then two AND-connected properties can be introduced:

$$\overset{\times}{q}a \;\; \varepsilon \;\; \sigma(c) \qquad \text{and} \qquad \overset{\times}{\square}b \;\; \varepsilon \;\; \sigma(b)$$

We notice that the complications that have already appeared with the handling of defined symbols (i.e. AND-connected verification properties, and rules of the type "for each object symbol b, ... ") re-appear in the handling of non-chaining inference rules. However, there are no new complications. This means that the non-chaining rules do not disturb the structure of the verification procedure. In fact, a corresponding statement holds for the rejection procedure.

11. Conclusion.

This report has resulted in the following:

(1) Specification of a logical language for use in property strctures (e.g. property lists). The language can handle binary relations, universal and existential quantifiers, " ε -quantifiers", and some implications;

(2) Inference rules for this language (I1-I9, J1-J7, etc.);

(3) Proof procedures ("rejection" and "verification" procedures) which are sound with respect to all inference rules, and complete with respect to a subset of the inference rules. These procedures perform (in principle) a search on an AND-OR tree, which is a relatively well-known type of search.

These results are interesting together but not taken independently. Our motivation for studying a property structure oriented language was to find a notation where inference can be performed efficiently, which means we must present a proof procedure together with the language. Similarly, the proof procedures are based on an associativity condition, so they are not interesting unless we can give a language which satisfies this condition. - These are the reasons why the above three results have been presented in the same paper. It is also an excuse for the lack of detail in this paper: we have tried to outline the results and their interrelationship. It is expected that later papers should concentrate on each of the three results, and go deeper into the details. The following tasks remain to be done:

(1) Extend the language, at least to include "systems of definitions"

11. Conclusion.

This report has resulted in the following:

(1) Specification of a logical language for use in property strctures (e.g. property lists). The language can handle binary relations, universal and existential quantifiers, " ε -quantifiers", and some implications;

(2) Inference rules for this language (I1-I9, J1-J7, etc.);

(3) Proof procedures ("rejection" and "verification" procedures) which are sound with respect to all inference rules, and complete with respect to a subset of the inference rules. These procedures perform (in principle) a search on an AND-OR tree, which is a relatively well-known type of search.

These results are interesting together but not taken independently. Our motivation for studying a property structure oriented language was to find a notation where inference can be performed efficiently, which means we must present a proof procedure together with the language. Similarly, the proof procedures are based on an associativity condition, so they are not interesting unless we can give a language which satisfies this condition. - These are the reasons why the above three results have been presented in the same paper. It is also an excuse for the lack of detail in this paper: we have tried to outline the results and their interrelationship. It is expected that later papers should concentrate on each of the three results, and go deeper into the details. The following tasks remain to be done:

(1) Extend the language, at least to include "systems of definitions"

(see section 9). Then describe the richness of the property structure language, e.g. by specifying a subset L' of the set of all wff in predicate calculus, such that each formula in L' can be expressed in property structure language, and vice versa.

(2) Give a complete set of inference rules for the extended language, together with a proof of completeness.

(3) Extend the proof procedures and prove their completeness.

However, we do not have to wait for these extensions and completeness proofs before we start to use the language. The material given in this paper is believed to be quite sufficient for a useful question-answering system.

References.

Ginsburg 1966a — S Ginsburg
The mathematical theory of context-free languages
McGraw-Hill

Green 1968a — C C Green, B Raphael
The use of theorem-proving eechniques in question-answering systems
Paper at 1968 ACM Conference, Las Vegas

Levien 1965a — R Levien, M E Maron
Relational Data File: A tool for mechanized inference execution and data retrieval
RM-4793-PR (RAND Corp, Santa Monica, Cal.)

Lindsay 1962a — R K Lindsay
A program for parsing sentences and making inferences about kinship relations
Proceedings of Western Management Science Conference on Simulation (A Hoggatt, ed.)

Raphael 1964a — B Raphael
SIR - a computer program for semantic information retrieval
MIT math dept., Ph D thesis, 1964

Robinson 1965a	J A Robinson A machine oriented logic based on the resolution principle Journal of the ACM, January, 1965
Sandewall 1968c	E J Sandewall LISP A: A LISP-like system for incremental computing Proc. Spring Joint Computer Conf., 1968
Sandewall 1968d	E J Sandewall Concepts and methods for heuristic search Uppsala University, Computer Sciences Dept., Report nr 16
Slagle 1963a	J R Slagle A heuristic program that solves symbolic integration problems in freshman calculus in E A Feigenbaum (ed), Computers and Thought
Slagle 1965b	J R Slagle A proposed preference strategy using sufficiency-resolution for answering questions UCRL-14361 (Lawrence Radiation Labs, Cal.)
Slagle 1968a	J R Slagle, Ph Bursky Experiments with a multi-purpose, theorem-proving heuristic program in Journal of the ACM, January, 1968