# 8

# A Note on Inductive Generalization

Gordon D. Plotkin
Department of Machine Intelligence and Perception
University of Edinburgh

In the course of the discussion on Reynolds' (1970) paper in this volume, it became apparent that some of our work was related to his, and we therefore present it here.

R.J.Popplestone originated the idea that generalizations and least generalizations of literals existed and would be useful when looking for methods of induction. We refer the reader to his paper in this volume for an account of some of his methods (Popplestone 1970).

Generalizations of clauses can also be of interest. Consider the following induction:

The result of heating this bit of iron to 419°C was that it melted.
The result of heating that bit of iron to 419°C was that it melted.
The result of heating any bit of iron to 419°C is that it melts.

We can formalize this as:

Bitofiron (bit 1) $\wedge$ Heated (bit 1,419) $\supset$ Melted (bit 1)
Bitofiron (bit 2) $\wedge$ Heated (bit 2,419) $\supset$ Melted (bit 2)
$(x)$ Bitofiron $(x)$ $\wedge$ Heated $(x, 419)$ $\supset$ Melted $(x)$

Note that both antecedents and conclusion can be expressed as clauses in the usual first-order language with function symbols. Our aim is to find a rule depending on the form of the antecedents which will generate the conclusion in this and similar cases. It will turn out that the conclusion is the least generalization of its antecedents.

We say that the literal $L_1$ is more general than the literal $L_2$ if $L_1\sigma = L_2$ for some substitution $\sigma$. For clauses we say that the clause $C_1$ is more general than the clause $C_2$ if $C_1$ subsumes $C_2$. Although the implication relationship might give interesting results, the weaker subsumption relationship allows a more manageable theory. For example, we do not even know whether implication between clauses is a decidable relationship or not.

A least generalization of some clauses or literals is a generalization which is less general than any other such generalization. For example, $P(g(x), x)$ is a least generalization of $\{P(g(a()), a()), P(g(b()), b())\}$ and $P(g(x), x)$

is a least generalization of $\{Q(x) \vee P(g(a()), a()), R(x) \vee P(g(b()), b())\}$. We give another, more complex example taken from a board game situation later in the paper. Our logical language is that of Robinson (1965). MacLane and Birkhoff (1967) is a good reference for our algebraic language.

## PRELIMINARIES

We will use the symbols $t, t_1, u, \ldots$ for terms, $L, L_1, M, \ldots$ for literals, $D, D_1, D, \ldots$ for clauses, $\varphi$ for a function symbol or a predicate symbol or the negation sign followed by a predicate symbol.

A *word* is a literal or a term. We will use the symbols $V, V_1, W, \ldots$ for words.

We denote sequences of integers, perhaps empty, by the symbols $I, J, \ldots$.

We say that $t$ is in the $I$th place in $W$ iff:

when $I = \langle\rangle$, $t = W$ or

when $I = \langle i_1, \ldots, i_n \rangle$, then $W$ has the form $\varphi(t_1, \ldots, t_m)$ and $i_1 \leqslant m$ and $t$ is in the $\langle i_2, \ldots, i_n \rangle$th place in $t_{i_1}$. For example, $x$ is in the $\langle\rangle$th place in $x$, the $\langle 2 \rangle$th place in $g(y, x)$ and in the $\langle 3, 2 \rangle$th place in $P(a, b, g(y, x))$.

Note that $t$ is never in the $\langle\rangle$th place in $L$. We say that $t$ is *in* $W$, if $t$ is in the $I$th place in $W$ for some $I$.

$W_1 \leqslant W_2$ (read '$W_1$ is more general than $W_2$') iff $W_1\sigma = W_2$ for some substitution $\sigma$. For example, $P(x, x, f(g(y))) \leqslant P(l(3), l(3), f(g(x)))$. We can take $\sigma = \{l(3)|x,\ x|y\}$.

$C_1 \leqslant C_2$ (read '$C_1$ is more general than $C_2$') iff $C_1\sigma \subseteq C_2$ for some substitution $\sigma$. $C_1 \leqslant C_2$ means that $C_1$ subsumes $C_2$ in the usual terminology. For example, $P(x) \vee P(f()) \leqslant P(f())$. We can take $\sigma = \{f()|x\}$.

In both cases, the relation $\leqslant$ is a quasi-ordering. We have chosen to write $L_1 \leqslant L_2$ rather than $L_1 \geqslant L_2$ as Reynolds (1970) does, because in the case of clauses, $\leqslant$ is 'almost' $\subseteq$. Further, if $L^1$ is the universal closure of $L$ and $L^2$ is the element of the Lindenbaum algebra corresponding to $L^1$, then we have $L_1 \leqslant L_2$ iff $L_1^1 \rightarrow L_2^1$ iff $L_1^2 \leqslant L_2^2$.

## WORDS

Suppose that we can show that a property holds for variables and constants, and that whenever it holds for $t_1, \ldots, t_n$ then it holds for $\varphi(t_1, \ldots, t_n)$. Then the property holds for all words. This method of proof is called induction on words.

We write $W_1 \sim W_2$ when $W_1 \leqslant W_2$ and $W_2 \leqslant W_1$. As $\leqslant$ is a quasi-ordering, this defines an equivalence relation. It is known that $W_1 \sim W_2$ iff $W_1$ and $W_2$ are alphabetic variants.

Two words are *compatible* iff they are both terms or have the same predicate letter and sign.

If $K$ is a set of words, then $W$ is a *least generalization* of $K$ iff:

1. For every $V$ in $K$, $W \leqslant V$.
2. If for every $V$ in $K$, $W_1 \leqslant V$, then $W_1 \leqslant W$.

It follows from 2 that if $W_1$, $W_2$ are any two least generalizations of $K$, then $W_1 \sim W_2$.

We can define also the least generalization as a product in the following category. The objects are the words and $\sigma$ is a morphism from $V$ to $W$ iff $V\sigma = W$ and $\sigma$ acts as the identity, $\varepsilon$, on variables not in $V$. $V$ is then a least generalization of $\{W_1, W_2\}$ iff it is a product of $W_1$ and $W_2$. We could also have defined the dual of the least generalization. This would just be the most general unification of Robinson (1965) and would be the coproduct in the above category. This approach was suggested in a personal communication by R. M. Burstall, but we have not followed it up to any degree.

*Theorem* 1

Every non-empty, finite set of words has a least generalization iff any two words in the set are compatible.

Let $W_1$, $W_2$ be any two compatible words. The following algorithm terminates at stage *3*, and the assertion made there is then correct.

*1.* Set $V_i$ to $W_i (i=1, 2)$. Set $\varepsilon_i$ to $\varepsilon (i=1, 2)$. $\varepsilon$ is the empty substitution.

*2.* Try to find terms $t_1$, $t_2$ which have the same place in $V_1$, $V_2$ respectively and such that $t_1 \neq t_2$ and either $t_1$ and $t_2$ begin with different function letters or else at least one of them is a variable.

*3.* If there are no such $t_1$, $t_2$ then halt. $V_1$ is a least generalization of $\{W_1, W_2\}$ and $V_1 = V_2$, $V_i\varepsilon_i = W_i (i=1, 2)$.

*4.* Choose a variable $x$ distinct from any in $V_1$ or $V_2$ and wherever $t_1$ and $t_2$ occur in the same place in $V_1$ and $V_2$, replace each by $x$.

*5.* Change $\varepsilon_i$ to $\{t_i|x\}\varepsilon_i (i=1, 2)$.

*6.* Go to *2*.

*Example.* We will use the algorithm to find a least generalization of

$$\{P(f(a(), g(y)), x, g(y)), P(h(a(), g(x)), x, g(x))\}.$$

Initially,

$$V_1 = P(f(a(), g(y)), x, g(y))$$
$$V_2 = P(h(a(), g(x)), x, g(x)).$$

We take $t_1 = y$, $t_2 = x$ and $z$ as the new variable. Then after *4*,

$$V_1 = P(f(a(), g(z)), x, g(z))$$
$$V_2 = P(h(a(), g(z)), x, g(z))$$

and after *5*,

$$\varepsilon_1 = \{y|z\}, \varepsilon_2 = \{x|z\}.$$

Next, we take $t_1 = f(a(), g(z))$, $t_2 = h(a(), g(z))$ and $y$ as the new variable. After *4* and *5*,

$$V_1 = P(y, x, g(z)) = V_2$$
$$\varepsilon_1 = \{f(a(), g(z))|y\}\{y|z\}$$
$$= \{f(a(), g(y))|y, y|z\}$$
$$\varepsilon_2 = \{h(a(), g(z))|y\}\{x|z\}$$
$$= \{h(a(), g(x))|y, x|z\}.$$

The algorithm then halts with $P(y, x, g(z))$ as the least generalization.

*Proof.* Evidently the compatibility condition is necessary. Let $\{W_1, \ldots, W_n\}$ be a finite compatible set of words. If $n=1$, then the theorem is trivial. Suppose that the algorithm works and that $\inf\{V, W\}$ is the result of applying it to $V$ and $W$. Then it is easy to see that

$$\inf\{W_1, \inf\{W_2, \ldots, \inf\{W_{n-1}, W_n\}\ldots\}\}$$

is a least generalization of the set. Hence we need only show that the algorithm works.

The rest of the proof proceeds as follows. In order to avoid a constant repetition of the conditions on $t_1, t_2$ given in 2, we say that $t_1$ and $t_2$ are *replaceable* in $V_1$ and $V_2$ iff they fulfil the conditions of *2*.

To show that the algorithm halts and that when it does $V_1=V_2$, we define a *difference* function by *difference* $(V_1, V_2)$ = number of members of the set $\{I|$*if* $t_1, t_2$ *are both in the Ith place in* $V_1, V_2$ *respectively then they are replaceable in* $V_1$ *and* $V_2\}$. We also denote by $V_1', V_2'$ the result of replacing $t_1$ and $t_2$ in $V_1, V_2$ by $x$ in the way described in *4*.

Lemma 1.2 then shows that every time a pair of replaceable terms is replaced the difference drops. Consequently by lemma 1.1 it will eventually become zero and when it does, lemma 1.1 shows that we must have $V_1=V_2$ and the algorithm will then halt.

We still have to show that the replacements take us in the correct direction. First of all, $V_i' < V_i$ since by lemma 1.3, $V_i'\{t_i|x\}=V_i$. It is also immediate from this that when the algorithm halts, $V_i'\varepsilon_i=W_i$. Now suppose that $W$ is any lower bound of $\{W_1, W_2\}$. Then a lower bound $V$ is a product of $W_1, W_2$ if the diagram of figure 1 can always be filled in along the dotted line, so that it becomes commutative in a unique way.

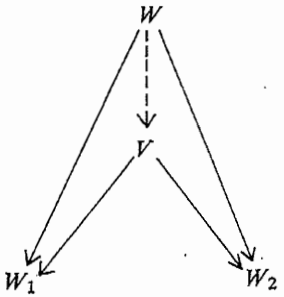



Figure 1

The category is the one defined above. In it there is either one or no morphisms between any two objects and hence it is not necessary in figure 1 to name the morphisms. Indeed, if a diagram can be filled in at all, it can be filled in commutatively and uniquely.

We show in lemma 1.4 that the diagram on figure 2 can be filled in commutatively.

Thus every time a replacement is made, the $V_i'$ are greater than any lower bound of $W_1$, $W_2$. Consequently when the algorithm halts, we have a product. We now give the statements and proofs of the lemmas.

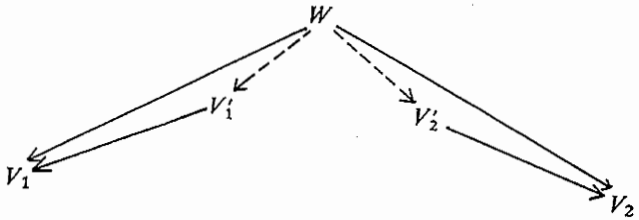



Figure 2

*Lemma* 1.1

If $V_1$ and $V_2$ are distinct compatible words, then there are $t_1$, $t_2$ which are replaceable in them.

*Proof.* By induction on words on $V_1$. If one of $V_1$, $V_2$ is a constant or a variable, or if they begin with different function symbols, then $V_1$, $V_2$ will do for $t_1$, $t_2$ respectively.

If $V_1$ is $\varphi(t_1^1, \ldots, t_n^1)$ and $V_2$ is $\varphi(t_1^2, \ldots, t_n^2)$, then for some $i$, $t_i^1 \neq t_i^2$ and by the induction hypothesis, applied to $t_i^1$, there are $u_1$, $u_2$ which are replaceable in $t_i^1$, $t_i^2$ and as $t_i^1$, $t_i^2$ have the same place in $V_1$, $V_2$ respectively, they are also replaceable in $V_1$, $V_2$.

*Lemma* 1.2

If $V_1$, $V_2$ are distinct compatible words, then *Difference* $(V_1', V_2') <$ *Difference* $(V_1, V_2)$.

*Proof.* By induction on words on $V_1$. If one of $V_1$ or $V_2$ is a variable or a constant then $t_1 = V_1$, $t_2 = V_2$ and $V_1' = V_2' = x$, so $0 =$ *Difference* $(V_1', V_2') < 1 =$ *Difference* $(V_1, V_2)$.

If $V_1$ is $f(v_1, \ldots, v_n)$ and $V_2$ is $g(u_1, \ldots, u_m)$ where $f \neq g$, then if $t_i = V_i (i = 1, 2)$, $0 =$ *Difference* $(V_1', V_2') <$ *Difference* $(V_1, V_2)$, by lemma 1.1; otherwise,

$$\textit{Difference}(V_1', V_2') = 1 + \sum_{i=1,\min(m,n)} \textit{Difference}(v_i', u_i')$$

$$< 1 + \sum_{i=1,\min(m,n)} \textit{Difference}(v_i, u_i)$$

(by induction hypothesis, since $m, n \neq 0$)

$$= \textit{Difference}\ (V_1, V_2).$$

In the remaining case where $V_1$ and $V_2$ both have the form $\varphi(t_1, \ldots, t_n)$, a similar but less complicated argument applies.

*Lemma* 1.3

$$V_i'\{t_i|x\} = V_i\ (i = 1, 2).$$

*Proof.* Since $V_i'$ is obtained from $V_i$ by replacing some occurrences of $t_i$ in $V_i$ by $x$, and since $x$ does not occur in $V_i$, substituting $t_i$ for $x$ in $V_i'$ will produce $V_i$. $(i = 1, 2)$.

*Lemma* 1.4

If $V_1$, $V_2$ are distinct compatible words and $V\sigma_i = V_i (i=1, 2)$, then there are $\sigma_1', \sigma_2'$ so that $V\sigma_i' = V_i' (i=1, 2)$.

*Proof.* It is convenient to denote by $f_i(u_1, u_2, t_1, t_2)$ the result of applying the operation of *4* to $u_1, u_2$ on $u_i (i=1, 2)$.

Let $\sigma_i = \{u_i^1 | y_1, \ldots, u_i^m | y_m\} (i=1, 2);$

$v_i^j = f_i(u_1^j, u_2^j, t_1, t_2) (i=1, 2; j=1, m);$

$\sigma_i' = \{v_i^1 | y_1, \ldots, v_i^m | y_m\} (i=1, 2).$

By lemma 1.3, $\sigma_i = \sigma_i' \{t_i | x\} (i=1, 2)$. We show by induction on $V$, that: if $V, V_1, V_2$ are such that $V\sigma_i = V_i (i=1, 2)$, then $V\sigma_i' = V_i' (i=1, 2)$.

Suppose that $V$ is a constant, then $V = V_1 = V_2$ and the result is trivial. Suppose that $V$ is a variable, $y$. If $y \neq y_i$ for $i=1, m$ then $y = V = V_1 = V_2$ and the result is again trivial. If $y = y_i$ say, then $V\sigma_i' = f_i(V_1, V_2, t_1, t_2) = V_i'$. Suppose $V$ is $\varphi(u_1, \ldots, u_n)$ then if $V_i = \varphi(w_1^i, \ldots, w_n^i)$,

$$V\sigma_i' = \varphi(u_1\sigma_i', \ldots, u_n\sigma_i') = \varphi(w_1^i, \ldots, w_n^i) \text{ (by the induction hypothesis)} = V_i'.$$

This concludes the proof.

The next lemma is used in the proof of the existence of least generalizations of clauses.

*Lemma* 2

Let $K = \{W_i | i=1, n\}$ be a set of words with a least generalization $W$ and substitutions $\mu_i (i=1, n)$ so that $W\mu_i = W_i (i=1, n)$.

*1.* If $t$ is in $W$ then $t$ is a least generalization of $\{t\mu_i | i=1, n\}$.

*2.* If $x, y$ are variables in $W$ and $x\mu_i = y\mu_i (i=1, n)$ then $x = y$.

*Proof. 1.* Evidently, $t$ is a generalization of $\{t\mu_i | i=1, n\}$. Suppose $u$ is any other and that $u\lambda_i = t\mu_i (i=1, n)$. Let $u\tau$ be an alphabetic variant of $u$ such that $u\tau$, $W$ have no common variables. Let $W'$ be $W$, but with $t$ replaced by $u\tau$ wherever $t$ occurs in $W$. Then $\tau^{-1}\lambda_i \cup \mu_i$ is defined – this follows from the construction of $\tau$ – and $W'(\tau^{-1}\lambda_i \cup \mu_i) = W_i (i=1, n)$. Hence there is a $v$ so that $W'v = W$, as $W$ is a least generalization of $\{W_i | i=1, n\}$. Hence $u(\tau v) = (u\tau)v = t$. Hence, $t$ is a least generalization of $\{t\mu_i | i=1, n\}$.

*2.* Suppose that $y \neq x$. Let $W' = W\{y | x\}$. Then $W'$, $W$ are not alphabetic variants, but $W \leqslant W'$. Let $W = W[x, y, y_3, \ldots, y_m]$, where $x, y, y_3, \ldots, y_m$ are the distinct variables of $W$. We have,

$$\begin{aligned} W_i = W\mu_i &= W[x\mu_i, y\mu_i, y_3\mu_i, \ldots, y_m\mu_i] \\ &= W[y\mu_i, y\mu_i, y_3\mu_i, \ldots, y_m\mu_i] \text{ (by hypothesis)} \\ &= W[y, y, y_3, \ldots, y_m]\mu_i \\ &= W'\mu_i \text{(by construction)}. \quad (i=1, n). \end{aligned}$$

This contradicts the fact that $W$ is a least generalization of $\{W_i | i=1, n\}$. Hence $y = x$.

This completes the proof.

## CLAUSES

Just as we did with words, we write $C \sim D$, when $C \leqslant D$ and $D \leqslant C$. This defines an equivalence relation. We also say that $C$ is a least generalization of a set of clauses, $S$, when:

*1.* For every $E$ in $S$, $C \leqslant E$.

*2.* If for every $E$ in $S$, $D \leqslant E$, then $D \leqslant C$.

Any two least generalizations of $S$ are equivalent under $\sim$. However, when $C_1 \sim C_2$, $C_1$ and $C_2$ need not be alphabetic variants. For example, take

$$C_1 = \{P(x), P(f())\}; \quad C_2 = \{P(f())\}.$$

It turns out that there is a reduced member of the equivalence class, under $\sim$, of any clause. This member is unique to within an alphabetic variant. $C$ is *reduced* iff $D \subseteq C$, $D \sim C$ implies that $C = D$. In other words, $C$ is reduced iff it is equivalent to no proper subset of itself.

*Lemma* 3

If $C\mu = C$, then $C$ and $C\mu$ are alphabetic variants.

*Proof.* We regard $C$ as a set ordered by $\leqslant$, and suppose without loss of generality that $\mu$ acts as the identity on variables not in $C$. Let $L$ be in $C$. The sequence $L = L\mu^0$, $L\mu^1 = L\mu$, $L\mu^2$, ... is increasing relative to $\leqslant$. As $C$ is finite and all members of the sequence are in $C$, it follows that for some $i$, $L\mu^i \sim L\mu^j (j \geqslant i)$. Hence for some $N$ and for all $L$ in $C$, $L\mu^N \sim L\mu^{N+i} (i \geqslant 0)$. As $C\mu = C$, there is an $M$ in $C$ so that $M\mu^N = L$, given $L$ in $C$. Hence, $L = M\mu^N \sim M\mu^{N+1} = L\mu$, and so $\mu$ maps variables into variables. But as $C\mu = C$, $C$ and $C\mu$ have the same number of variables. Hence $\mu$ maps distinct variables of $C$ to distinct variables of $C\mu$, and so $C$ and $C\mu$ are alphabetic variants, thus completing the proof.

*Theorem* 2

If $C \sim D$, and $C$ and $D$ are reduced, then they are alphabetic variants. The following algorithm gives a reduced subset, $E$, of $C$ such that $E \sim C$.

*1.* Set $E$ to $C$.

*2.* Find an $L$ in $C$ and a substitution $\sigma$ so that $E\sigma \subseteq E \setminus \{L\}$.

If this is impossible, stop.

*3.* Change $E$ to $E\sigma$ and go to *2.*

(It is necessary to be able to test for subsumption in order to carry out stage *2.* Robinson (1965) gives one way to do this.)

*Proof.* As $C \sim D$, there are $\mu, \nu$ so that $C\mu \subseteq D$, $D\nu \subseteq C$. Hence, $C\mu\nu \subseteq C$. But $C$ is reduced so $C\mu\nu = C$. Hence by lemma 3, $\mu\nu$ maps the variables of $C$ into the variables of $C$ in a 1–1 manner. Hence $C$ and $D$ are alphabetic variants.

The algorithm halts at stage *2*, since the number of literals in $E$ is reduced by at least one at stage *3* and so if it does not halt before then, it will halt when this number is 1. If $C = \emptyset$, then it will halt on first entering *2*.

There is always a $\mu$ so that $C\mu \subseteq E$. For at stage *1*, take $\mu = \varepsilon$. If one has such

a $\mu$ before stage 2, then $\mu\sigma$ will be one after stage 2. Hence, when the algorithm halts, $C\mu \subseteq E$ and $E \subseteq C$, and then $C \sim E$.

Suppose $E$ is not reduced at termination. Then there is a proper subset $E'$ of $E$ so that $E' \sim E$. So there is a $\sigma$ such that $E\sigma \subseteq E'$ .Pick $L$ in $E \setminus E'$. Then $E\sigma \subseteq E' \subseteq E \setminus \{L\}$. This contradicts the fact that the algorithm has terminated and completes the proof.

This theorem is useful as our method of producing least generalizations of clauses tends to give clauses with many literals which may often be substantially reduced by the above procedure.

Let $S = \{C_i | i = 1, n\}$ be a set of clauses.

A set of literals, $K = \{L_i | i = 1, n\}$ is a *selection* from $S$ iff $L_i \in C_i (i = 1, n)$.

We can now state the main theorem.

*Theorem 3*

Every finite set, $S$, of clauses has a least generalization which is not $\emptyset$ iff $S$ has a selection. If $C_1$ and $C_2$ are two clauses with at least one selection, then the following algorithm gives one of their least generalizations.

Let $S = \{C_1, C_2\}$, and let the selections from $S$ be $\{L_l^1, L_l^2\} (l = 1, n)$ where $L_l^j$ is in $C_j$. Suppose that $L_l^j = (\pm)P_l(t_{1l}^j, \ldots, t_{k_l l}^j)$, where $(\pm)P$ is either $P_l$ or $\bar{P}_l$. Let $f_l$ be a function letter with $k_l$ places $(l = 1, n)$, and let $P$ be a predicate letter with $n$ places. Let $M_j$ be the literal

$$P(f_1(t_{11}^j, \ldots, t_{k_1 1}^j), \ldots, f_n(t_{1n}^j, \ldots, t_{k_n n}^j)) \; (j = 1, 2).$$

Find the least generalization of $\{M_1, M_2\}$ by the method of theorem 1. Suppose that this is

$$M = P(f_1(u_{11}, \ldots, u_{k_1 1}), \ldots, f_n(u_{1n}, \ldots, u_{k_n}))$$. Let $C$ be the clause

$$\{(\pm P)_i(u_{1i}, \ldots, u_{k_i i}) | i = 1, n\}.$$

Then $C$ is a least generalization of $S = \{C_1, C_2\}$.

Evidently, it is not necessary in any actual calculation to change any $P_l$ to the corresponding $f_l$.

*Proof.* We begin by showing that the algorithm works. By theorem 1, there are $\nu_i (i = 1, 2)$ so that $M\nu_i = M_i$. From lemma 2, it follows that $f_i(u_{1i}, \ldots, u_{k_i i})$ is a least generalization of $\{f_i(t_{1i}^j, \ldots, t_{k_i i}^j) | j = 1, 2\}$.

Hence $L_i = (\pm)P_i(u_{1i}, \ldots, u_{k_i i})$ is a least generalization of $\{L_i^1, L_i^2\} (i = 1, n)$.

Hence $C\nu_i = \bigcup_{l=1}^{n} \{L_l^i\} \subseteq C_i$ and so $C$ is a generalization of $\{C_1, C_2\}$.

Note also that by lemma 2, if $x\nu_i = y\nu_i (i = 1, 2)$, then $x = y$.

Now suppose that $E$ is any generalization of $\{C_1, C_2\}$. We show that $E \leqslant C$. Let $\alpha_l$ be chosen so that $E\alpha_l \subseteq C_i$. $(i = 1, 2)$, and let $E = \{M_1, \ldots, M_m\}$. $\{M_p\alpha_i | i = 1, 2\}$ will be a selection, say $\{L_{p'}^1, L_{p'}^2\}$. Consequently, $M_p \leqslant L_{p'}$, the corresponding least generalization of the selection, and there is a $\beta_p$ so that $M_p\beta_p = L_{p'}$.

Hence, if $\overset{m}{\underset{p=1}{\cup}}\beta_p$ exists, $E(\cup\beta_p)\subseteq C$ and we will have finished. Now $\cup\beta_p$ exists precisely if, whenever $x$ is an individual variable in $M_{p_1}$ and $M_{p_2}$ $(p_1\neq p_2)$ then $x\beta_{p_1}=x\beta_{p_2}$. Let the notation $A\overset{\alpha}{\rightarrow}B$ ($A$, $B$ literals; $\alpha$ a substitution) mean $A\alpha=B$. We have shown that the relationships described by figure 3 hold:

$(i,k=1,2)$

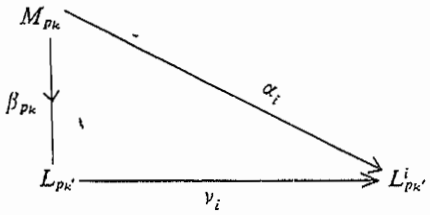



Figure 3

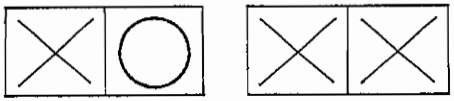
Now $x\beta_{p_k}$ is a term in $L_{p_k'}(k=1,2)$, which is a least generalization of $\{L^1_{p_k'}, L^2_{p_k'}\}$. Hence, by lemma 2, $x\beta_{p_k}$ is a least generalization of $\{x\beta_{p_k}\nu_1, x\beta_{p_k}\nu_2\}=\{x\alpha_1, x\alpha_2\}$ from the diagram, $(k=1,2)$.

Consequently, $x\beta_{p_1}, x\beta_{p_2}$ are alphabetic variants. Let $x_1, x_2$ be variables having the same place in $x\beta_{p_1}, x\beta_{p_2}$ respectively. Then as $x\beta_{p_1}\nu_i=x\beta_{p_2}\nu_i=x\alpha_i(i=1,2)$, it follows that $x_1\nu_i=x_2\nu_i(i=1,2)$. Hence by our note on the properties of the $\nu_i(i=1,2)$ at the beginning of the proof, $x_1=x_2$. Hence $x\beta_{p_1}=x\beta_{p_2}$, and $\cup\beta_p$ exists and $E\leqslant C$ and so $C$ is a least generalization of $\{C_1, C_2\}$.

Next, we note that $C$ is not empty and indeed if a particular combination of sign and predicate letter occurs in a selection from $\{C_1, C_2\}$ then it occurs in $C$. Suppose that $S=\{C_i\}(i=1,q)$ is a finite set of clauses. If $C(\neq\emptyset)$ is a generalization of $S$, there are $\alpha_i$ so that $C\alpha_i\subseteq C_i$, and if $L$ is in $C$, $\{L\alpha_i|i=1,q\}$ is a selection. On the other hand, $\emptyset$ is a generalization of $S$. Hence if $S$ has no selection its only, and hence its least, generalization is $\emptyset$. Otherwise, by nesting infs as in the proof of theorem 1, we can find a least generalization of $S$ which, by the note at the beginning of this paragraph, will not be $\emptyset$. This completes the proof.

## AN EXAMPLE

Suppose that some two-person game is being played on a board with two squares, 1() and 2() and that the positions in figure 4 are won positions for the first player:



Position $p_1()$ Position $p_2()$

Figure 4

1() is the name of the left hand side square, and 2() of the right hand square; $p_1()$ and $p_2()$ are the names of the positions and $O()$, $X()$ are the names of the marks $O$, $X$. We describe the fact that these positions are wins by means of the following two clauses:

1. $\overline{Occ}(1(), X(), p_1()) \vee \overline{Occ}(2(), O(), p_1()) \vee Win(p_1())$.
2. $\overline{Occ}(1(), X(), p_2()) \vee \overline{Occ}(2(), X(), p_2()) \vee Win(p_2())$.

The course of the calculation is indicated as follows:

$\overline{Occ}(1(), X(), p_1())\ \overline{Occ}(1(), X(), p)$
$\overline{Occ}(1(), X(), p_2())\ \overline{Occ}(1(), X(), p)$

$\overline{Occ}(1(), X(), p_1())\ \overline{Occ}(1(), X(), p)\ \overline{Occ}(n_1, X(), p)$
$\overline{Occ}(2(), X(), p_2())\ \overline{Occ}(2(), X(), p)\ \overline{Occ}(n_1, X(), p)$

$\overline{Occ}(2(), O(), p_1())\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(n_2, O(), p)\ \overline{Occ}(n_2, x, p)$
$\overline{Occ}(1(), X(), p_2())\ \overline{Occ}(1(), X(), p)\ \overline{Occ}(1(), X(), p)\ \overline{Occ}(n_2, X(), p)\ \overline{Occ}(n_2, x, p)$

$\overline{Occ}(2(), O(), p_1())\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(2(), x, p)$
$\overline{Occ}(2(), X(), p_2())\ \overline{Occ}(2(), X(), p)\ \overline{Occ}(2(), X(), p)\ \overline{Occ}(2(), X(), p)\ \overline{Occ}(2(), x, p)$

$Win(p_1())$ $\quad$ $Win(p)$
$Win(p_2())$ $\quad$ $Win(p)$

As indicated above, we have not replaced predicate symbols by function symbols, and we have left $P$ implicit. Each vertical column displays $M_1$ and $M_2$ at an instance of stage *2* of the algorithm of theorem 1. In a given column, the pairs of literals are corresponding arguments in $M_1$ and $M_2$. We find $t_1$ and $t_2$ by searching through $M_1$ and $M_2$ from left to right. As soon as two literals have become the same in a column, we do not mention them in subsequent columns.

Thus the least generalization is:

$\overline{Occ}(1(), X(), p) \vee \overline{Occ}(n_1, X(), p) \vee \overline{Occ}(n_2, x, p) \vee \overline{Occ}(2(), x, p) \vee Win(p)$.

We use the algorithm of theorem 2. We can take $L = \overline{Occ}(n_1, X(), p)$, $\sigma = \{1()|n_1\}$. This gives

$C\sigma = \overline{Occ}(1(), X(), p) \vee \overline{Occ}(n_2, x, p) \vee \overline{Occ}(2(), x, p) \vee Win(p)$.

Next, we can take $L = \overline{Occ}(n_2, x, p)$ and $\sigma = \{2()|n_2\}$ and obtain

$C\sigma = \overline{Occ}(1(), X(), p) \vee \overline{Occ}(2(), x, p) \vee Win(p)$.

The algorithm stops at this point. The final clause says that if a position has an $X$ in hole 1 and hole 2 has something in it, then the position is a win, which, given the evidence, is fairly reasonable.

We leave it to the reader to verify that the conclusion of the inductive argument given at the beginning of this note is indeed a least generalization of its antecedents. The main computational weakness in the method for finding a reduced least generalization lies in that part of the reducing algorithm which requires a test for subsumption. For suppose that we are looking for the inf of two clauses each with nine literals in a single predicate letter (this can arise in descriptions of tic–tac–toe, say); there will be at least eighty-one literals in the raw inf, and we will have to try to tell whether or not a clause of eighty-one literals subsumes one of eighty.

## FURTHER RESULTS

We give without proof some further results obtained on the algebraic nature of the $\leqslant$ relation between clauses.

Let $[C]$ denote the equivalence class under $\sim$ of $C$. We say that $[C] \leqslant [D]$ iff $C \leqslant D$. It is easily seen that this is a proper definition.

The set of equivalence classes forms a lattice with the lattice operations given by:

$$[C] \sqcap [D] = [\inf \{C, D\}]$$
$$[C] \sqcup [D] = [C\xi \cup D],$$

where $\xi$ is a substitution which standardizes the variables of $C$ and $D$ apart.

This lattice is not modular. It has an infinite strictly ascending chain $[C_i]$ $(i \geqslant 1)$ where

$$C_1 = \{P(x_0, x_1)\}; \quad C_i = C_{i-1} \cup \{P(x_{i-1}, x_i)\}.$$

This chain is bounded above by $\{P(x, x)\}$.

There is also a rather complicated infinite strictly descending chain, $[C_i]$ $(i \geqslant 1)$ with the following properties:

*1.* No literal in any $C_i$ contains any function symbols.

*2.* The clauses are all formed from a single binary predicate letter and a single unary one.

Any infinite descending chain is bounded below by $\emptyset$. We hope to publish the proofs elsewhere.

**Acknowledgements**

I should like to thank my supervisors R.J. Popplestone and R.M. Burstall for all the different kinds of help they gave me. Particular thanks are due to Dr Bernard Meltzer, without whose encouragement this paper would not have been written. The work was supported by an SRC research studentship.

## REFERENCES

MacLane, S. & Birkhoff, G. (1967) *Algebra.* New York: Macmillan.

Popplestone, R.J. (1970) An experiment in automatic induction. *Machine Intelligence 5* pp. 203–15 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Reynolds, J.C. (1970) Transformational systems and the algebraic structure of atomic formulas. *Machine Intelligence 5* pp. 135–52 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Robinson, J.A. (1965) A machine-oriented logic based on the resolution principle. *J. Ass. comput. Mach.*, **12**, 23–41.