PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

APPLYING EXPLANATION-BASED GENERALIZATION TO NATURAL-LANGUAGE PROCESSING

Manny Rayner

Swedish Institute of Computer Science, Box 1263
S-164 28 KISTA, Sweden

ABSTRACT

It is shown how ideas adapted from recent work on explanation-based generalization can be used to allow a logic grammar to "learn" useful derived grammar rules by generalizing them from example sentences. The method is presented in the form of a small Prolog meta-interpreter, and its soundness is formally proved. Examples are given showing the application of the generalizer, first to a toy grammar with 40 rules and then to a largish independantly developed system which involves non-trivial syntactic and semantic analysis.

1. INTRODUCTION

Everyone who has tried to build a serious natural-language interface will be aware of a certain fundamental tradeoff. On the one hand, it is desirable to implement syntactic and semantic rules that are as general as possible. This is good for a multitude of reasons: it makes for a flexible and portable system, it gives a clear declarative reading of program code, and it can uncover results which throw light on theoretical linguistic issues. However, there is a catch: a system implemented in this way tends to make exorbitant demands on time and space resources. Even if the underlying parsing mechanism is made as efficient as possible, the grammar ends up running like cold molasses because so many unlikely dead ends are being explored.

Let us look at a couple of examples. Conjunction is a very general phenomenon, and most theoreticians agree that the best way to deal with it is by using some sort of treatment which captures the underlying uniformity of conjunction rules for various syntactic categories. However, it's usually a bit of a waste of time to look for a word like "neither" or "both" when you want to parse an adverb; very few adverbs are conjoined forms like "neither slowly nor stupidly" or "both quickly and cleverly". Similarly, a fronted NP at the begining of a WH-question is seldom modified by a relative clause[1], as in a question like "Which people *you* know would agree to that?"; an attempt to find one will normally be an expensive failure.

Moreover, it is a matter of common experience that different domains make different demands on the grammar. A construction common in one domain may hardly exist in another, and ignoring this fact can also turn out to have expensive consequences. The outcome of all this is that the interface implementor is more or less forced to "tune" his grammar with respect to the practical necessities of the problem. This can be done in various ways: rules can be merged together and tests inserted in strategic places to aid efficiency, or else can simply be eliminated if the application sublanguage does not appear to need them (Grishmann et al 84). People are starting to get some idea of how to do this systematically, but it's still an expensive and messy process that takes time and specialized expertise. It would be highly desirable to be able to perform this task automatically.

A promising first step in this direction is reported in (Ramsay 85). Working in a GPSG framework, Ramsay discusses the problem of knowing how far meta-rules ought to be expanded before they are used. Not expanding them at all makes it difficult to parse efficiently, and expanding everything is out of the question due to space limitations; Ramsay's ingenious solution to the dilemma is to give the grammar a set of example sentences, and let it learn from these which rule expansions occur frequently. These can then be added explicitly to the grammar to give quick recognition of common types of phrase.

The key operation in Ramsay's idea is that of constructing a "generalized" version of the specific rule expansion from the example sentence. Unfortunately, his description of this step is extremely sketchy; I quote (p.59):

[1] I am indebted to Michael McCord for this observation.

```
%A Prolog meta-interpreter for Hirsh-style EBG. The predicates operational_goal/1,
%insufficiently_instantiated/1 and built_in_goal/1 are domain-dependant, and
%should be supplied by the user.

generalize(Goal,General_goal,Conds) :- generalize_1(Goal,General_goal,[],Conds).

generalize_1(true,true,Conds,Conds) :- !.

generalize_1((H,T),(G_H,G_T),In_conds,Out_conds) :- !,
       generalize_1(H,G_H,In_conds,Next_conds),
generalize_1(T,G_T,Next_conds,Out_conds).

generalize_1((H;T),(G_H;G_T),In_conds,Out_conds) :- !,
       (generalize_1(H,G_H,In_conds,Out_conds);
        generalize_1(T,G_T,In_conds,Out_conds)).

generalize_1(Goal,G_goal,In_conds,Out_conds) :-
       built_in_goal(Goal),
       (insufficiently_instantiated(G_goal),!, Out_conds = [G_goal|Conds];
        Out_conds = In_conds, call(G_goal)),
       call(Goal).

generalize_1(Goal,G_goal,In_conds,[G_goal|In_conds]) :-
       operational_goal(Goal),!,call(Goal).

generalize_1(Goal,G_goal,In_conds,Out_conds) :-
       expand_goal(Goal,G_goal,Body,G_body),
generalize_1(Body,G_body,In_conds,Out_conds).

expand_goal(Goal,G_goal,Body,G_body) :-
       functor(Goal,P,N), functor(Head,P,N),
       clause(Head,Body), copy_term([Head,Body],[G_head,G_body]),
       Goal = Head, G_goal = G_head.
```

Diagram 1

```
s(np(cat,the,[3,s],A,[])                ;The subject is a 3rd-person singular noun-phrase with main noun cat,
                                        ;determiner the, index A and no modifiers.
  vp(verb(see,[3,s],imperfect,pos);The main verb of the verb-phrase is the 3rd-person singular
    [virtual np: A                      ;imperfect positive form of see. The filler for its agent case is a
     np(dog,the,[3,s],B,[])]            ;virtual copy of the subject, and that for its object case is the
     virtual np: A))                    ;noun-phrase np(dog...). The VP is controlled by the subject
```

Diagram 2a

```
Sentence [A,B,C,D,E]

has parse-tree:

s(np(F,A,[3,G],H,[])                    ;The subject is a 3rd-person, number G noun-phrase with main noun F,
                                        ;determiner A, index H and no modifiers.
  vp(verb(I,[3,G],J,pos)                ;The main verb of the verb-phrase is the 3rd-person number G tense J
   [virtual np: H                       ;positive form of I. The filler for its agent case is a virtual copy of
     np(K,D,[3,L],M,[])]                ;the subject, and that for its object case is
    virtual np: H))                     ;the noun-phrase np(K,D...). The VP is controlled by the subject

IF                                      ;assuming the following conditions are fulfilled:

lex(A,det)                              ;The first word, A, is a determiner
lex(B,noun(F,G))                        ;The second, B is a form of noun F, number G
lex(C,verb(I,J,3,G,[agent,object]));The third, C is a form of verb I, tense J 3rd-G,
                                        taking agent and object cases.
lex(D,det)                              ;The fourth, D, is a determiner
lex(E,noun(K,L))                        ;The fifth, E, is a form of noun K, number L
```

Diagram 2b

... we keep a record of text fragments that we have previously managed to analyze. When we make an entry in this record, we abstract away from the text the details of exactly which words were present. What we want is a general description of them in terms of their lexical categories, features such as transitivity, and endings ... Alongside each of them we keep an abstracted version of the structure that was found, i.e. of the parse tree that was constructed to represent the way we did the analysis. Again, the abstraction is produced by throwing away the details of the actual words that were present, replacing them this time by indicators saying where in the original text they appeared.

What I am going to do here is to rework Ramsay's idea in the context of a logic grammar. I will show that it is then possible, using recent ideas from explanation-based learning theory, to give an exact and rigorous characterization of the generalization operation. This characterization will moreover cover not only syntactic, but also semantic processing.

2. EXPLANATION-BASED GENERALIZATION IN A LOGIC GRAMMAR

Let's look at Ramsay's idea from the point of view of logic programming. A logic grammar is a Prolog program; at various points, we can make non-deterministic choices as to which clause to resolve with next. Some of these choices correspond to rule selection in a top-down parser; others might correspond to (for example) selection of different scoping orders in the semantic component.

We assume that we have managed to process an example sentence: that is, we have found a set of non-deterministic choices which can be used to construct a proof that the input string is a well-formed sentence with an associated logical form. Now we want to generalize our result. Following the ideas described in (Hirsh 87), we start by dividing the set of predicates in the system into two subsets, which we refer to as *operational* and *non-operational* respectively; the idea is that we are going to try and abstract away the information contributed by the "operational" predicates. Thus if we are to follow Ramsay's ideas as quoted above, we might wish to define as operational those predicates which define the part of speech of a word, the number and person of a verb, and whether or not a noun is animate.

Continuing our programme of translating Hirsh's ideas into a natural-language context, we work through the parse of the example sentence, keeping track of all the operational calls. We do this by performing two computations in parallel, the second, "generalized" one being "slaved" to the first in the sense that all rule applications are decided by it. When an operational predicate is encountered, the first computation resolves against it in the normal way; the second one, however, succeeds without performing any resolution, and saves the operational call in a list which we will refer to as the *operational condition stack*.

When the computations terminate, the second one will be a generalization of the first; it will consist of a proof that the conjunction of the literals on the operational condition stack imply the initial goal, after it has been subjected to the various substitutions that have been generated in the course of the proof.

In the interests of providing a runnable specification of what we have just said, we present the procedure in the form of a one-page Prolog meta-interpreter (see diagram 1); if the program we are optimizing is "clean" (i.e. contains no cuts, unsound negations as failure, or extra-logical predicates like "nonvar"), this is moreover provably correct. This is a good argument in favour of clean logic grammars.

Readers concerned with the formal aspects of the matter are at this point referred to the appendix, where the generalization process is described in mathematical terms and its soundness proved. Others (the majority, I suspect), who are willing to take soundness on trust, should read on. I start by giving a couple of examples showing the idea in action: the grammar used is a toy logic grammar of about 40 rules, written in XG notation (Pereira 83). Initially, the only operational predicate is `lex/2`, which constitutes the interface to the lexicon.

Example 1

The input sentence we will use in our first example is

```
[the,cat,saw,the,dog]
```

We can parse the sentence, getting the syntax-tree in Diagram 2a. The output from the generalizer is then shown in 2b:

Example 2

This is similar, though slightly more complicated. The input sentence is

```
[the,man,that,bought,the,cat,has,a,dog]
```

the parse-tree for the specific sentence is

```
s(np(man,the,[3,s],A,
     [s(virtual np: A
        vp(verb(buy,[3,s],imperfect,pos)
            [virtual np: A
             np(cat,the,[3,s],B,[])]
            virtual np: A))])
  vp(verb(have,[3,s],present,pos)
     [virtual np: A
      np(dog,a,[3,s],C,[])]
     virtual np: A))
```

and the output from the generalizer is

```
Sentence [A,B,C,D,E,F,G,H,I]

has parse-tree:

s(np(J,A,[3,K],L,
     [s(virtual np: L
         vp(verb(M,[3,s],N,pos)
            [virtual np: L
             np(O,E,[3,P],Q,[])]
            virtual np: L))]])
  vp(verb(R,[3,K],S,pos)
     [virtual np: L
      np(T,H,[3,U],V,[])]
     virtual np: L))

IF

lex(A,det)
lex(B,noun(J,K))
lex(C,rel_pro)
lex(D,verb(M,N,3,s,[agent,object]))
lex(E,det)
lex(F,noun(O,P))
lex(G,verb(R,S,3,K,[agent,object]))
lex(H,det)
lex(I,noun(T,U))
```

3. VARIANTS ON THE BASIC SCHEME

3.1 GOING BEYOND SYNTAX

We now want to explore the generalization idea and see what more we can do with it. Note first, that, in contrast to Ramsay's original formulation, we are in no way limited to just generalizing over the syntax. It is quite possible to generalize over the whole path from input string to logical form, and indeed beyond; the only limitation comes from the fact that we have to work with "clean" logic programs. But for the moment, let's assume that we want to go no further than to the logical form.

The non-deterministic choice-points in the semantic phase are primarily going to be the scoping transformations. In most systems, including ours, these are driven by determiner and case information, so generalization preserves a lot. This is apparent in example 3, below. The only unclear point arises from the well-known trick by which λ-bound variables in the λ-calculus are identified with uninstantiated logical variables (this is discussed at length in (Warren 83)). Warren rightly refers to this as "an efficiency hack to use Prolog's built-in variable-handling facilities to speed the λ-reduction"; there are potential problems due to the use of the "var" and "identity" (==) meta-predicates. However, my experimental findings so far seem to indicate that this is probably not serious. If it is, it is anyway always possible to rewrite the grammar cleanly in the way indicated by Warren, at the cost of a small overhead in efficiency and perspicuity.

Example 3

This example is considerably more advanced than the first two, and illustrates generalization over semantic processing operations in the Swedish grammar for comparatives from (Banks & Rayner 87), (Rayner & Banks 88). The grammar had to be rewritten slightly to make it "clean"; this involved about a day's work for a system which contains about 80 XG rules and 140 other clauses. (See digram 4)

This is as far as I have gone to date in my practical experimentation. However, in a system with non-trivial pragmatic processing it may be possible to go even further; thus a story-understanding system may well be able to produce generalized versions of updates from examples showing how a specific example sentence is used to update a discourse structure, if suitable discourse primitives are deemed to be operational. This seems to me to be an interesting idea to explore further.

3.2 CHANGING THE "GRAIN-SIZE"

So far, we have implicitly assumed that "operational" predicates are essentially going to be those that look up words in the lexicon: to express it in another way, the units we are generalizing over are going to be words. This is however by no means necessary, since the scheme will work just as well irrespective of which predicates are defined as operational; one obvious candidate for operationality is "NP". As can be seen in examples 4 and 5 below, choosing "NP" as operational produces generalizations where the NP's are regarded as primitive objects. One attractive idea is to use this to produce a "two-layer" grammar; the "top" or "outer" layer is an NP-primitive grammar, while the "lower" or "inner" one is a grammar for NP's produced in exactly the same way. This would give a system which has certain similarities with Marcus parsing (Marcus 80). Other variants are certainly conceivable, and the only way to find the optimum one is presumably to experiment.

Example 4

We redo example 1, this time with `np` defined as operational as well. As before, the input sentence is

```
[the,cat,saw,the,dog]
```

The specific result is the same as in example 1. This time, however, the output from the generalizer is as shown in diagram 5:

```
[vilka,kungar,f|ddes,under,samma,}rhundrade,som,Karl XII]

Which   kings   were-born  during  the-same  century        as    Karl XII?

wh_q(wh_plural(A
                [rel,is_a,A,kung]                                          ;A is a king
                ex(B
                    and([[rel,is_a,B,}rhundrade]                           ;B is a century
                        the(C
                            and([[rel,not_equal,C,A]                       ;C ≠ A
                                 [rel,name_of,C,Karl XII]])                ; C 's name is "Karl XII"
                            ex(D
                                and([[rel,is_a,D,}rhundrade]               ;B is a century
                                     [rel,samma,B,D]])                     ;B is the same as D
                                ex_event(E
                                        and([[rel,is_a,E,f|da]             ;event E is of type "be born"
                                             [rel,object,E,C]              ;the object of E is C
                                             [rel,during,E,D]])))))]);E occurs during D
                    ex_event(F
                            and([[rel,is_a,F,f|da]                         ;event F is of type "be born"
                                 [rel,object,F,A]                          ;the object of F is A
                                 [rel,during,F,B]]))))))                   ;F occurs during B
General result:

Sentence [vilka,A,B,C,D,E,F,G] has logical form:

wh_q(wh_plural(K
                [rel,is_a,K,J]
                ex(T
                    and([[rel,is_a,T,S]
                        the(R
                            and([[rel,not_equal,R,K],[rel,name_of,R,Q]])
                            ex(U
                                and([[rel,is_a,U,S],[rel,M,T,U]])
                                ex_event(V
                                        and([[rel,is_a,V,H]
                                             [rel,object,V,R]
                                             [rel,L,V,U]])))))])
                    ex_event(W
                            and([[rel,is_a,W,H],[rel,object,W,K],[rel,L,W,T]])))))

IF

word(A,J,noun(undet,plur))                           ;A is the undetermined plural form of the noun J
question_article(vilka)                              ;vilka is a question article (this could be removed)
word(B,H,verb(passive,I,[agent(X),object(Y)]))   ;B is a passive form of the transitive verb H, whose
                                                     ; agent and object fillers have selectional
                                                     ;restrictions to types X and Y
fits_type(J,Y)                                       ;J is of semantic type Y
word(C,Z,preposition)                                ;C is a form of the preposition Z
word(D,M,article(comparative,identity,F,O,P))        ;D is a form of the identity-comparing determiner M,
                                                     ;whose associated complementizer is F and whose
                                                     ;number and determination are O and P
word(E,S,noun(O,P))                                  ;E is a form of the noun S ,whose number and
                                                     ;detemination are O and P
dif(F,null)                                          ;the complementizer F is non-null
word(G,Q,name)                                       ;G is a form of the name Q
preposition_interpretation(Z,L,A1)                   ;the preposition Z can be interpreted as marking the
                                                     ;case L ,if it occurs with an NP with semantic type A1
fits_type(S,A1)                                      ;S is of semantic type A1
```

Diagram 4

5. SUMMARY

I have described an implementation of Hirsh's work on explanation-based generalization in the form of a small Prolog meta-interpreter, and proved its correctness. I have then applied this to the problem of finding "generalized" versions of interpretations of natural-language sentences, and shown that the method is practically usable in non-trivial contexts.

Having worked for some time on a project aimed at the construction of a large-scale natural-language interface (Rayner & Banks 86), (Rayner & Banks 88b), it seems to me the ideas described here have a very promising future. Although we have no statistical evidence to support our claim, our experience from demonstrating the system is that the same *sort* of questions turn up time and time

```
Sentence A has parse-tree:

s(np(B,C,[D,E],F,H)                     ;The subject is a Dth-person, number E noun-phrase with main noun B,
                                        ;determiner C, index F and modifiers H.
 vp(verb(H,[D,E],I,pos)                 ;The main verb of the verb-phrase is the
  [virtual np: H                        ;Dth-person number E tense I positive form of H. The filler for its
   J]                                   ;agent case is a virtual copy of the subject, and that for its
                                        ;object case is the noun-phrase J.
   virtual np: H))                      ;The VP is controlled by the subject


IF

np(np(B,C,[D,E],F,G),A,[K|L],[],[]);We can parse np(B,C,[D,E],F,G),
                                        ;starting with A and ending with [K|L].
lex(K,verb(H,I,D,E,[agent,object]));K is a form of verb H, tense I, agr [D,E],
                                        ;taking agent and object cases.
np(J,L,[],[],[])                        ;We can parse the NP J, starting with L and ending with [].


It is easier to see what this means if we write it down in XG notation as the derived grammar
rule

s(np(B,C,[D,E],F,G),vp(verb(H,[D,E],I,pos),[virtual np:F,J],virtual np: F)) -->
      np(np(B,C,[D,E],F,G),[K],{lex(K,verb(H,I,D,E,[agent,object]))},np(J).
```

Diagram 5

again; I can easily believe that the fifty most common question types could account for more than 95% of all queries submitted to a given database. The generalization method, which can automatically identify and "compile" these common question-types, would thus promise enormous efficiency gains at a negligable cost.

In conclusion, and at risk of straying a little too far from the subject in hand, I wish to state my conviction that most natural-language researchers have so far paid far too little attention to machine-learning. NL is an area where the effects of the "knowledge acquisition bottleneck" are close to being omnipresent; in the long run, I do not think that an NL system with human competence can be constructed in any other way than by letting it infer rules from examples according to very general schemas. Whether or not the ideas presented here will play any part in the development of such a system is, of course, a question for the future.

ACKNOWLEDGEMENTS

I would like to thank Lars Asker, Amelie Banks, Göran Hagert and Annika Wærn for useful comments and criticism.

REFERENCES

(Banks & Rayner 87) Banks, A. and Rayner, M., Comparatives in Logic Grammars - Two Viewpoints, pp. 153-169, V. Dahl & P. St.Dizier (eds.) *Natural Language Understanding and Logic Programming II*, North-Holland1987

(Grishman et al 84) Grishman, R., Nhan, N.T., Marsh, E. and Hirschmann, L., Automated Determination of Sublanguage Usage, pp. 96-100, *Proc. 22nd COLING*, Stanford, 1984

(Hirsh 87) Hirsh, H., Explanation-Based Generalization in a Logic-Programming Environment, pp. 221-227, *Proc. 10th IJCAI*, pp. 221-227, Milano, 1987

(Marcus 80) Marcus, M., *A Theory of Natural Language Parsing*, Ph.D. thesis, MIT, 1980

(Pereira 83) Pereira, F.N.C. *Logic for Natural Language Analysis*, SRI Technical Note No 275, 1983

(Ramsay 85) Ramsay, A., Effective parsing with Generalized Phrase-Structure Grammar, pp. 57-61, *Proc. 2nd European ACL*, Geneva 1985

(Rayner & Banks 86) Rayner, M. & Banks, A. Temporal Relations and Logic Grammars, pp.9-14, Vol. 2, *Proc. 7th ECAI*, 1986

(Rayner & Banks 88) Rayner, M. & Banks, A., Parsing and Interpreting Comparatives, to appear in *Proc. 26th ACL*, Buffalo, 1988

(Rayner & Banks 88b) Rayner, M. & Banks, A., *An Overview of SNACK-85*, SICS research report, 1988

(Robinson 79) Robinson, J.A., *Logic - Form and Function*, Edinburgh University Press, 1979

(Warren 83) Warren, D.S., Using λ-Calculus to Represent Meanings in Logic Grammars, pp. 51-56, *Proc. 21st ACL*, MIT, 1983

APPENDIX: SOUNDNESS OF THE GENERALIZATION OPERATION

We need the following preliminary definitions:

Definition 1
A *goal-identifier* is a finite sequence of non-negative integers.

We will use . to signify the concatenation of two sequences, thus e.g. <1,3>.<4,5,6> is <1,3,4,6,5>. ε will represent the empty sequence, and ι the trivial substitution.

Definition 2
A *goal-literal* is a pair <G,I>, where G is a literal and I a goal-identifier.

Definition 3
A *goal-set* is a finite set of goal-literals.

Definition 4
If G and G' are goal-sets, and Γ is a set of Horn clauses, then a *labelled one-step SLD derivation of G' from G using Γ, with associated substitution θ* is defined as follows:

i) If G = G', then **Null** is a labelled one-step SLD derivation of G from G' using Γ, with associated substitution ι.

ii) If the following holds:

a) $<L,I> \in G$.
b) $C = (H \leftarrow B_1 \ldots B_n) \in \Gamma$ such that L and H are unifiable with mgu θ.
c) G' is the set $\theta(G - <L,I>) \cup \theta\{<B_1, I.<1>>, <B_2, I.<2>> \ldots <B_n, I.<n>>\}$

then <I,C> is a labelled one-step SLD derivation of G' from G using Γ, with associated substitution θ. We will call <L,I> the *selected goal-literal*, L the *selected literal*, and the predicate symbol in the head of L the *selected predicate* for <I,C>. It will also be useful to refer to the greatest common instance of two literals S and T as S+T.

Definition 5
If G and G'' are goal-sets, and Γ is a set of Horn clauses, then a *labelled SLD derivation of G'' from G using Γ, with associated substitution θ* is recursively defined as follows:

i) If G = G'', then ε is a labelled SLD derivation of G from G'' using Γ, with associated substitution ι.
ii) If the following holds:

a) Σ is a one-step labelled SLD derivation of G' from G using Γ with associated substitution θ.
b) Δ is a labelled SLD proof of G'' from G' using Γ with associated substitution ϕ.

then $<\Sigma>.\Delta$ (the sequence formed by adding Σ to the front of Δ) is a labelled SLD proof of G'' from G using Γ, with associated substitution $\phi\theta$.

Definition 6
If G, G' are goal-sets, then the *formula associated with* <G,G'> is defined to be the formula $\forall x_1 \ldots x_n (\Phi \rightarrow \Psi)$, where Φ is the conjunction of the literals occuring in G, Ψ is the conjunction of the literals occurring in G', and $x_1 \ldots x_n$ are the variables which occur free in Φ, Ψ.

The following is a trivial consequence of the soundness of resolution [Robinson 79]:

Theorem 1
Let G and G'' be goal-sets, and Γ a logic program. If there is a labelled SLD derivation of G'' from G using Γ with associated substitution θ, and Φ is the formula associated with <G'', θ(G)>, then $\Gamma \Rightarrow \Phi$.

Definition 7
Let Γ be a logic program, and Δ be a labelled SLD derivation from $\varnothing$ using Γ. Suppose further that Π is the set of predicates in Γ, and that Λ is a subset of Π. Then gen(Δ,Λ), the *generalization of Δ over Λ*, is recursively defined as follows:

i) If $\Delta = \varepsilon$, then gen(Δ,Λ) = ε.
ii) If Δ = <**Null**>+Δ' for some Δ', then gen(Δ,Λ) = <Null>+gen(Δ',Λ)
iii) If Δ = <I,C>+Δ' for some I,C,Δ', and the selected predicate for <I,C> is an element of Λ, then gen(Δ,Λ) = <Null>+gen(Δ',Λ).
iv) If Δ = <I,C>+Δ' for some I,C,Δ', and the selected predicate for <I,C> is not an element of Λ, then gen(Δ,Λ) = <I,C>+gen(Δ',Λ).
Intuitively, gen(Δ,Λ) "misses out" all the steps in which a goal is resolved against a clause whose head is in Λ. The point of "labelling" SLD derivations is to be able to specify what this means in formal terms.

Definition 8
Let G and G' be goal-sets, and let Λ be as in the previous definition. Then $G <_\Lambda G'$ (pronounced: G is less instantiated than G' discounting Λ) iff there exist a substitution σ and a goal-set L such that:

i) The predicate symbol of each literal in L is a member of Λ.
ii) $\sigma(G' - L) = G$

It is now possible to state the result we wish to prove:

Theorem 2: (Soundness of generalization)
Let Γ be a logic program and G a goal-set, and Δ be a labelled SLD derivation of G from $\varnothing$ using Γ with associated substitution ϕ. Let Λ and gen(Δ,Λ) be as in definition 7 above. Then given any goal-set G' such that $G <_\Lambda G'$, there is a goal-set L which satisfy the following conditions:

i) gen(Δ,Λ) is a labelled SLD derivation of G' from L using Γ.
ii) The predicate symbol in each goal of L is a member of Λ.

Proof
We use induction on the length of Δ. Suppose first that $\Delta = \varepsilon$. Then gen(Δ,Λ) = ε as well, so we can take $\theta = \iota$ and $L = \varnothing$ to conclude the proof.

Suppose now that $\Delta = <\Sigma>.\Delta'$ and assume that the hypothesis holds for all derivations shorter than Δ, and thus in particular for Δ'. There are three cases:

i) Σ = Null. Then gen(Δ,Λ) = <Null>.gen(Δ',Λ). Δ' is shorter than Δ, and is also a derivation of G. So by the induction hypothesis gen(Δ',Λ) is a suitable derivation for G', and thus gen(Δ,Λ) is also a suitable derivation for G'.

ii) Σ = <I,C> for some I,C, and the selected predicate for <I,C> is an element of Λ. Then gen(Δ,Λ) = <Null>.gen(Δ',Λ); we know that there are some σ and L such that $G = \sigma(G' - L)$. Let S be the selected goal-literal in G, and θ the associated substitution for <I,C>. Then Δ' is a labelled SLD derivation for $\theta(G - \{S\})$. We have that $\theta(G - \{S\}) = \theta\sigma(G' - (L \cup \{S\}))$, and thus $\theta(G - \{S\}) <_\Lambda G'$ since the predicate symbol of S is in Λ. By the induction hypothesis, gen(Δ',Λ) is a suitable labelled SLD-derivation of G', and so gen(Δ,Λ) is one too.

iii) Σ = <I,C> for some I,C, and the selected predicate for <I,C> is not an element of Λ. This is the non-trivial case. We have gen(Δ,Λ) = <<I,C>>.gen(Δ',Λ); as in ii) above, let σ and L be such that $G = \sigma(G' - L)$, S be the selected goal-literal in G, and θ the associated substitution for <I,C>. Let H and B be the head and body of C, let H' and B' be copies related by a variable-renaming substitution τ with inverse τ', and let S' be the goal-literal in G' whose identifier is I. Assume further that variables in C and its copy are "unique", i.e. do not occur in G or G'.

We have that $S = \sigma S'$. We know that S and H are unifiable with mgu θ; thus S' and H' must also be unifiable. Call the mgu of S' and H θ'. Since there are substitutions $\theta\sigma{:}S' \rightarrow S+H$ and $\theta\tau{:}H' \rightarrow S+H$, there is a σ' such that the following diagram commutes[1]:

$$\begin{tikzcd}
S' \arrow[dd, "\sigma"'] \arrow[dr, "\theta'"] & & H' \arrow[dl, "\theta'"'] \arrow[dd, shift right, "\tau"'] \\
& S'+H' \arrow[dd, "\sigma'"] & \\
S \arrow[dr, "\theta"'] & & H \arrow[dl, "\theta"] \arrow[uu, shift right, "\tau'"'] \\
& S+H &
\end{tikzcd}$$

We know that Δ' is a labelled SLD derivation for the goal-set resulting from the application of <I,C> to G, which by definition 4 above is $G_1 = \theta(G - \{S\}) \cup \theta\{<B_1, I.<1>>, <B_2, I.<2>>\ldots <B_n, I.<n>>\}$, B_i being the literals in B. The result of applying <I,C> to G' is $G_1' = \theta'(G' - \{S'\}) \cup \theta'\{<B'_1, I.<1>>, <B'_2, I.<2>>\ldots <B'_n, I.<n>>\}$. We now define the substitution σ'' as follows (since variables are unique, σ'' is well defined):

i) $\sigma''(x) = \sigma'(x)$ if x is a variable which occurs in S' or H'.
ii) $\sigma''(x) = \sigma(x)$ if x is a variable which occurs in G' but not in S'.
iii) $\sigma''(x) = \tau(x)$ if x is a variable which occurs in B' but not in H'.

Since θ and θ' only affect variables in S, S', H and H', we have $G_1' = \sigma''(G_1 - L)$, and thus $G_1' <_\Lambda G_1$. So by the induction hypothesis, gen(Δ',Λ) is a labelled SLD derivation for G_1' from a goal-set whose predicate symbols are in Λ, and thus gen(Δ,Λ) is one too. This concludes the proof.

[1] The reader should note the similarity to the "lifting lemma" ([Robinson 79], p.213).