Application of Explanation-Based Generalization in Theorem Proving*

TR89-015

March 1989

Xumin Nie
David A. Plaisted

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

This work was supported in part by the National Science Foundation under Grant #DCR-8516243 and by the Office of Naval Research under grant #N00014-86-K-0680

Application of Explanation-Based Generalization in Theorem Proving *

Xumin Nie *and* David A. Plaisted
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
Internet: {nie, plaisted}@cs.unc.edu
Telephone: {Nie: (919)-962-1734. Plaisted: (919)-962-1751}

Abstract

This paper presents a special case of Explanation-Based Generalization (EBG), Goal Generalization. Goal generalization, which tries to find a proof for the most general solvable version of a specific goal while solving the specific goal, is an application of EBG methods in automatic theorem proving and is a general technique applicable to many goal-oriented theorem proving systems. We will describe Goal Generalization as an augmentation for a sequent-style, goal-oriented theorem proving system for arbitrary quantifier-free clauses. Some implementation results are also given.

Key Words and Phrases: Explanation-Based Generalization, extension of Prolog to first order logic, automatic theorem proving.

Length in words. 4000.

1. Introduction and Motivation

Many theorem proving systems have a goal-oriented structure. They use backward chaining as their main inference mechanism. An example of such system is the SLD-resolution for Horn Clause Logic [Lloyd 84]. Other examples are [Loveland 88, Stickel 86, Plaisted 88]. Basically, a goal-oriented system starts to attempt a top-level G. A goal L will be declared as attempted if it is already solved based on the database of assertions the system maintains; otherwise, L will be decomposed into several goals $L_1, L_2, ..., L_n$ (there may be more than one ways to decompose L), which are called subgoals, and each of the subgoals will be attempted in the same manner.

A situation may arise that, for a goal-oriented system, a goal to be solved is very specific and a proof for a more general version of the goal exists and the proof for the more general goal has the same structure as the proof for the specific goal. Consider an example in Horn Clause Logic:

```
r(X) :- p(X), q(X).
p(X).
q(X).
```

The top-level goal is r(a). Obviously a proof exists for a more general goal r(X). It is beneficial to find the proof for the most general solvable version of a goal while the goal is being solved, especially when caching is performed, that is, the solutions to the goals are recorded and later used as assertions to solve other goals. For the example above, a goal r(b) will be declared as solved if

* This work was supported in part by the National Science Foundation under grant DCR-8516243 and by the Office of Naval Research under grant N00014-86-K-0680.

r(X) instead of r(a) is solved and cached. Repeated work is avoided.

To accomplish the task of finding proofs for the most general solvable version of a goal while solving the goal, we need some generalization capability in the theorem proving system. In this paper, we will discuss our research to add one such generalization capability to a particular theorem proving system based on the modified problem reduction format [Plaisted 88]. We will call our approach *goal generalization*. The basic idea is to augment the theorem proving system to keep two versions of the goals being attempted, one of which is the specific goal to be solved and the other is a possibly more general version of the specific goal. The proof constructed will be for the more general version and can be instantiated to become a proof for the specific goal.

Some related works will be discussed in section 2. In particular, we will formalize goal generalization as a special case of Explanation-Based Generalization. We will briefly describe the modified problem reduction format in section 3. In section 4, we will show how goal generalization is incorporated in the modified problem reduction format by augmenting the theorem proving system. We conclude the paper with some implementation results.

2. Related Works

Explanation-Based Generalization is a technique recently developed in the field of machine learning [Mitchell&al 86]. This technique deals with the problem of formulating general concepts on the basis of specific training examples. This technique has been shown to be the same as the technique in functional programming, Partial Evaluation, in [Harmelen&Bundy 88]. Goal generalization can be regarded as a special case of the Explanation-Based Generalization problem. In an Explanation-Based Generalization problem, we are given

- Goal concept — describing the concept to be learned;
- Training example — an example of the goal concept;
- Domain theory — a set of rules and facts about the domain;
- Operationality criterion — criterion for the form of the learned concept definition;

and are to determine a generalization of the training example that is a sufficient concept definition for the goal concept and that satisfies the operationality criterion.

We can reformulate goal generalization in terms of Explanation-Based Generalization as follows:

- Goal concept — the most general solvable version of the goal to be solved;
- Training example — the specific goal to be solved;
- Domain theory — all the inference rules;
- Operationality Criterion — the goal concept (a goal) must be a logical consequence of the domain theory (according to the inference rules).

Generally speaking, our work is one example of the research issues raised in [Mitchell&al 86]:

> ...how such methods for generalization will be used as subcomponents of larger systems that improve their performance at some given task. ...One key issue to consider in this regard is how generalization tasks are initially formulated. In other words, where do the inputs to the EBG method (the goal concept, the domain theory, the operationality criterion) come from?...

In our approach, the attempt to solve each goal is formulated as a generalization task and this formulation bears a task-subtask structure similar to the goal-subgoal structure of the theorem proving system; The input clause set serves as the domain theory; And the concept of logical consequence serves as the operationality criterion.

The idea of augmenting an existing theorem proving system to construct two proofs in parallel to achieve Explanation-Based Generalization is also used in [Kedar-Cabelli&McCarty 87], where Explanation-Based Generalization is presented as an augmentation of the SLD-resolution theorem proving system for Horn Clause Logic. Our approach extends this idea to full first-order logic by augmenting a theorem proving system for full first-order logic. We point out that our approach can be used for many other goal-oriented systems to accomplish a similar task.

3. Modified Problem Reduction Format

The modified problem reduction format is an extension of Prolog to full first-order logic (non-Horn clauses). The modified problem reduction format accepts a set of *Horn-like clause* as input. A *Horn-like clause* is of the form $L :- L_1, L_2, \cdots, L_n$, which represents the clause $L \vee \neg L_1 \vee \neg L_2 \cdots \neg L_n$, where L is called the *head literal* and $L_1, \cdots, L_n$ constitute the *clause body*. A general clause C is converted into a Horn-like clause HC as follows. One of the positive literal in C is chosen as the head literal of HC and all other literals in C are negated and put in the clause body of HC. If C contains only negative literals, we use the special literal FALSE as the head literal of HC.

The inference rules for the modified problem reduction format consist of the *clause rules*, which are obtained from the input clauses, the *assumption axioms* and the *case analysis rule*. For each Horn-like clause $L :- L_1, L_2, \ldots, L_n$ in S, we have a clause rule. We call the lists of literals Γ's on the left of the arrow $\rightarrow$ *assumption list*.

Clause Rules

$$\frac{[\Gamma_0 \rightarrow L_1 \Rightarrow \Gamma_1 \rightarrow L_1], [\Gamma_1 \rightarrow L_2 \Rightarrow \Gamma_2 \rightarrow L_2], \ldots, [\Gamma_{n-1} \rightarrow L_n \Rightarrow \Gamma_n \rightarrow L_n]}{\Gamma_0 \rightarrow L \Rightarrow \Gamma_n \rightarrow L}$$

The assumption axioms and the case analysis rule are

Assumption Axioms

$$\Gamma \rightarrow L \Rightarrow \Gamma \rightarrow L \ \text{ if } L \in \Gamma \quad L \text{ is a literal.}$$

$$\Gamma \rightarrow \neg L \Rightarrow \Gamma, \neg L \rightarrow \neg L \quad L \text{ is positive.}$$

Case Analysis Rule

$$\frac{[\Gamma_0 \rightarrow L \Rightarrow \Gamma_1, \neg M \rightarrow L], [\Gamma_1, M \rightarrow L \Rightarrow \Gamma_1, M \rightarrow L] \quad |\Gamma_0| \leq |\Gamma_1|}{\Gamma_0 \rightarrow L \Rightarrow \Gamma_1 \rightarrow L}$$

The implementation of the modified problem reduction format in [Plaisted 88] uses depth-first iterative deepening search [Korf 85] with caching and true unification (unification with occur-check). In this implementation, each inference rule is represented as a Prolog clause and a goal $\Gamma \rightarrow L$ is represented as $L :- \Gamma$. The main procedure is mprf(((L :- B_0), (L :- B_1)) where L :- B_0 is the goal to be solved and L :- B_1 is the goal solved; B_1 is B_0 probably with extra negative literals added to it at the front. The top-level call is mprf(((false :– [ ]), (false :– B)) and the solved goal being false :– [ ] indicates a successful proof. For an input clause L :– $L_1, L_2, \cdots, L_n$, the following Prolog clause represents the corresponding clause rule:

```
mprf((L0 :– B0), (L0 :– Bn)) :–
        unify(L0, L),
        mprf((L1 :– B0), (L1 :– B1)),
        ...
        mprf((Li :– Bi-1), (Li :– Bi)),
        ...
        mprf((Ln :– Bn-1), (Ln :– Bn)),
```

where L_0 is logical variable in Prolog. For a unit clause L, the corresponding clause rule is represented as

```
mprf((L0 :– B), (L0 :– B)) :– unify(L0, L).
```

where L_0 is logical variable in Prolog. The representations for the assumption axioms and the case analysis rule are

```
mprf((L :– B), (L :– B)) :– member(L, B).
mprf((not(L) :– B), (not(L) :– [not(L)|B])).

mprf((L :– B0), (L :– B1)) :–
        mprf((L :– B0), (L :– [not(M)|B1])),
        mprf((L :– [M|B1]), (L :– [M|B1])),
        length(B0) ≤ length(B1).
```

The procedure *unify* performs true unification. The procedure *member* is defined as follows

```
member(L, [X|Y]) :– unify(L, X).
member(L, [X|Y]) :– member(L, Y).
```

We have only provided a simplified description on the aspects of the modified problem reduction format and its implementation necessary for the subsequent discussion. Many details and subtleties about the inference system and the implementation are omitted for brevity. See [Plaisted 88] for a complete discussion.

4. Goal Generalization

If a call mprf((L :– B_0), (L :– B_1)) succeeds, the goal L :– B_1 has a proof. It is possible that a more general goal LG :– BG_1 has a proof with the same structure as that for L :– B_1. The more general goal LG :– BG_1 is a generalization in the sense that all goals that can be obtained from LG :– BG_1 by a substitution have proofs of the same structure. Goal generalization tries to find the most general solvable version LG :– BG of a goal L :– B while solving L :– B. We achieve this by augmenting the Prolog representation for the inference rules with extra arguments. Those extra arguments represent the more general versions of their counterparts. To be specific, the procedure mprf((L :– B_0), (L :– B_1)) will be replaced by mprf_GG((L :– B_0), (L :– B_1), (LG :– BG_0), (LG :– BG_1)), where L :– B_0 and L :– B_1 are the goal to be solved and the goal solved, respectively, as the two arguments in the procedure mprf are, and LG :– BG_0 and LG :– BG_1 are the more general versions of L :– B_0 and L :– B_1 respectively. The result is that a proof for LG :– BG_1 will be constructed which can be instantiated to be a proof for L :– B_1. The resulting representation for a clause rule for the Horn-like clause L :– $L_1, L_2, \cdots, L_n$ will be

```
mprf_GG((L0 :– B0), (L0 :– Bn),(LG0 :– G0), (LG0 :– Gn)) :–
```

unify(L_0, L),
variable_list(G_0, VL_0), make_var(LG_1, V_1),
mprf_GG((L_1 :- B_0), (L_1 :- B_n),(V_1 :- VL_0), (V_1 :- G_1)),
unify(V_1, LG_1), unify(VL_0, G_0),
. . .
variable_list(G_{i-1}, VL_{i-1}), make_var(LG_i, V_i),
mprf_GG((L_i :- B_{i-1}), (L_i :- B_i),(V_i :- VL_{i-1}), (V_i :- G_i)),
unify(V_i, LG_i), unify(VL_{i-1}, G_{i-1}),
. . .
variable_list(G_{n-1}, VL_{n-1}), make_var(LG_n, V_n),
mprf_GG((L_n :- B_{n-1}), (L_n :- B_n),(V_n :- VL_{n-1}), (V_n :- G_n)),
unify(V_n, LG_n), unify(VL_{n-1}, G_{n-1}),
unify(LG_0, LG).

where L_0 and LG_0 are logical variables and make_var(L_i, V_i) ($i = 1, 2, ..., n$) is such that V_i will be a distinct logical variable if L_i is a positive literal, a term not(W_i) with W_i being a variable if L_i is a negative literal. :- LG_1, LG_2, $\cdots$, LG_n is a copy of L :- L_1, L_2, $\cdots$, L_n (with new variables) made during the preprocessing, that is, when the Prolog clause is generated. The procedure *variable_list* assemblies a list of distinct variables from a list of literals. For example, a list [X_1, X_2, X_3] will be returned by *variable_list*, given a list of three literals [L_1, L_2, L_3]. A unit clause L will be transformed into

mprf_GG((L_0 :- B), (L_0 :- B), (LG_0 :- BG), (LG_0 :- BG)) :- unify(L_0, L), unify(LG_0, LG).

where L_0 and LG_0 are logical variables and LG is a copy of L made during the preprocessing. Similarly, we also have the corresponding Prolog clause representations for the assumption axioms and the case analysis rule:

mprf_GG((L :- B), (L :- B), (LG :- BG), (LG :- BG)) :- member(L, B, LG, BG).
mprf_GG((not(L) :- B), (not(L) :- [not(L)|B]), (not(LG) :- BG), (not(LG) :- [not(LG)|BG])).

mprf_GG((L :- B_0), (L :- B_1),(LG :- BG_0), (LG :- BG_1)) :-
mprf_GG((L :- B_0), (L :- [not(M)|B_1]), (LG :- BG_0), (LG :- [not(MG)|BG_1])),
variable_list([MG|BG_1], VL), make_var(L, V),
mprf_GG((L :- [M|B_1]), (L :- [M|B_1]), (V :- VL), (V :- VL)),
unify(VL, [MG|BG_1]), unify(V, LG), length(B_0) $\leq$ length(B_1).

The procedure *member* is defined as follows

member(L, [X|Y], LG, [XG|YG]) :- unify(L, X), unify(LG, XG).
member(L, [X|Y], LG, [XG|YG]) :- member(L, Y), member(LG, YG).

The following theorem formalizes what goal generalization accomplishes:

Theorem: Given a set of input clauses S, if the call mprf_GG((L :- B_0), (L :- B_1), (V :- VL), (V :- BG_1)) succeeds, where V is a variable or a term not(W) with W being a variable depending on whether L is a positive or negative literal and VL is a list of variables constructed from B_0 by replacing each literal in B_0 with a distinct variable, then the following are true:

(1) There exists a substitution θ such that (V :- BG_1)θ = (L :- B_1);

(2) $BG_1 \supset V$ is a logical consequence of S, where BG_1 is interpreted as a conjunction of the literals in it; and

(3) If there is a substitution π, a goal (G :- M) which has the same proof as (V :- BG_1) does,[1]

and $(G :- M)\pi = (V :- BG_1)$, then π only renames variables; in another words, $V :- BG_1$ is the most general goal with the same proof.

Proof. By induction on the size of the proof for $V :- BG_1$, where the size of a proof is the number of inference rules used to obtain the proof. □

5. Implementation and Experimental Results

We have modified a theorem prover based on the modified problem reduction format to incorporate the augmentation discussed above. There are several refinements in the implementation that deserve more elaboration. The first refinement concerns using the Prolog built-in unification in place of some calls to the procedure *unify*. The second refinement concerns how to eliminate unnecessary use of the splitting rule. The third refinement concerns how to handle repeated solutions.

Using Prolog built-in unification. We can replace the calls to *unify(R, L)* that involve the more general versions of the goals by $R = L$, which invokes Prolog built-in unification. It is well known that Prolog omits the occur-check in its unification for efficiency, and unification without occur-check is unsound [Plaisted 84]. In our case, however, we use true unification for the specific goals and the unification operations involving the more general versions of the goals are always performed after the unification operations on the specific goals succeed, thus are guaranteed to succeed. Therefore it is sufficient to use Prolog built-in unification on the more general goals. This refinement is important for the efficiency of the augmented prover.

Unnecessary case analysis. It is made possible by the augmentation to detect when some splitting literals are not used during the proof and thus redundant. In the more general version of a goal, LG :- BG, the assumption list BG starts to be a list of logical variables. The only place where these variables can bound to a literal is in the assumption axiom where the procedure *member* is called. If a call mprf_GG($(L :- B_0)$, $(L :- B_1)$, $(LG :- BG_0)$, $(LG :- BG_1)$) succeeds and there are still unbound variables in BG_1, we know that there are redundant literals in the assumption list. Two alternatives are available to handle this. We can either delete those variables from the assumption lists or simply fail on the call to mprf_GG. This is a potentially powerful deletion strategy and is made possible by the augmentation. This strategy seems to be similar to the requirement in Near-Horn Prolog that there be cancellation within each restart block in a legal deduction [Loveland 88]. In our implementation, we elect the option of failing on redundant literals in assumption lists.

Repeated solutions. We treat a solution R as a repeated solution if there is already a solution S in the database such that R is subsumed by S and the proof length of S is no greater than that of R (This is not quite correct theoretically, but seems to work well in practice). Since we are deriving and caching the most general solvable goals, it is more likely that repeated solutions are generated. In our implementation, we elect to fail when a repeated solution is generated based on the consideration that the search would be repeated if we succeed.

We have tested the augmented prover, with the three refinements discussed above, on the problem set from [Stickel 86]. We have made the original prover fail on repeated solutions too, in order to make a fair comparison. The table at the end shows our test result. We note that, in 72

[1] We say two goals $(L_1 :- B_1)$ and $(L_2 :- B_2)$ have the same proof if they use the same inference rules in the same order.

out of 82 problems, the augmented prover generates fewer or equal number of solutions for most problems and, for the 35 problems on which the augmented prover generates fewer solutions, the number of solutions is reduced by 38.6 percent on the average. This is one benefit we have expected by adding the generalization capability. This is probably one of the reasons why the augmented prover is faster on wos15. However, the inference rate of the augmented prover is much less than that of the original prover (3.63 inferences per second as opposed to 5.00 inferences per second), due to the extra arguments. This is why the augmented prover is much slower on problems like wos4, ls65 and schubert, where the numbers of solutions generated and the numbers of inferences performed by the original prover and by the augmented prover differ very little. We want to point out that, without the three refinements discussed above, the augmented prover performs poorly, and even fails to obtain proofs for some problems (wos31 and ls108).

References

[Harmelen&Bundy 88] Harmelen, F. and A. Bundy, "Explanation-Based Generalization = Partial Evaluation", *Artificial Intelligence* 36, pp. 401-412, 1988.

[Kedar-Cabelli&McCarty 87] Kedar-Cabelli, S.T. and L.T. McCarty, "Explanation-Based Generalization as Resolution Theorem Proving", In P. Lanley, editor, *Proceedings of the 4th International Machine Learning Workshop*, pp. 383-389, Morgan Kaufmann, 1987.

[Korf 85] Korf, R.E., "Depth-first Iterative Deepening: an Optimal Admissible Tree Search", *Artificial Intelligence*, Vol. 27, 97-109, 1985.

[Lloyd 84] Lloyd, J.W., *Foundations of Logic Programming*, New York , NY, Springer-Verlag.

[Loveland 88] Loveland, D., "Near-Horn Prolog and Beyond", Technical Report #CS-1988-25, Computer Science Department, Duke University, Durham, 1988.

[Mitchell&al 86] Mitchell, T.M., R.M. Keller and S.T. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View", *Maching Learning*, Vol. 1, No. 1, pp. 47-80, 1986.

[Plaisted 84] Plaisted, D.A., "The Occur-Check Problem in Prolog", *Journal of New Generation Computing* 2, pp. 309-322, 1984.

[Plaisted 88] Plaisted, D.A., "Non-Horn Clause Logic Programming Without Contrapositives", *Journal of Automated Reasoning*, Vol 4, No. 3, September 1988.

[Stickel 86] Stickel, M.E., "A PROLOG Technology Theorem Prover: Implementation by an Extended PROLOG Compiler", *Proc. of IJCAI*, pp. 573-587, Oxford, England, July 1986.

Test Result for Goal Generalization (GG)[2]								
	original prover				augmented prover			
theorem	proof depth	running time	number of inference	number of solution	proof depth	running time	number of inference	number of solution
ances1	18	2.98	22	7	18	4.20	22	8
burstall	11	4.97	103	15	11	7.98	103	2
dbabhp	11	8.60	163	51	11	13.47	163	51
dm	9	0.73	11	2	7	0.58	8	2
ew1	9	0.70	7	5	9	0.83	7	5
ew2	7	0.38	4	4	7	0.40	4	4
ew3	11	1.25	11	6	11	1.50	11	6
ex1	9	0.95	14	2	7	0.77	9	2
ex2	11	5.87	308	7	11	8.27	309	2
ex3	9	1.13	33	4	7	1.02	27	4
ex4	9	1.23	35	4	7	1.12	28	4
ex5	7	0.33	6	2	7	0.42	6	2
ex6	9	2.68	134	9	9	3.53	134	9
ex7	9	0.87	13	6	9	1.07	13	6
ex8	11	2.30	54	8	11	2.90	55	8
ex9	11	3.02	37	8	11	3.77	37	8
example	18	20.97	613	10	18	22.00	417	4
fex4t1	18	242.22	1033	196	18	383.58	1038	173
fex4t2	18	159.60	853	150	18	222.97	833	118
fex5	11	309.72	2967	297	11	1461.28	3710	318
fex6t1	24	26.93	935	27	18	35.72	881	36
fex6t2	24	25.20	895	24	18	37.88	887	31
group1	9	1.12	18	2	7	0.80	10	2
group2	11	5.90	308	7	11	7.80	309	2
hasparts1	11	1.45	24	6	11	2.05	25	6
hasparts2	24	4.30	81	11	24	5.90	80	10
ls100	7	0.40	6	3	7	0.50	7	3
ls103	18	6.67	115	9	18	10.18	116	5
ls105	7	0.70	11	4	7	0.75	11	4
ls106	7	0.67	11	4	7	0.75	11	4
ls108	24	375.07	3403	67	24	1358.47	3572	129
ls111	7	0.55	9	4	7	0.77	11	4
ls115	11	10.78	164	13	11	19.93	150	13
ls116	9	10.40	126	39	9	19.70	122	38
ls121	11	52.85	884	57	11	83.82	748	48
ls17	9	3.50	69	9	9	4.90	64	9
ls23	11	11.73	318	24	9	15.70	300	24
ls26	9	1.68	63	6	9	2.15	63	6
ls28	11	60.45	610	131	9	61.00	579	133
ls29	11	58.83	602	130	9	55.95	575	132
ls35	14	8.48	358	6	11	9.93	350	6
ls41	7	2.12	45	8	7	2.92	46	5
ls5	7	0.43	5	5	7	0.57	5	5
ls55	7	2.45	37	4	5	2.82	31	5

Test Result for Goal Generalization (GG) (Cont.)								
	original prover				augmented prover			
theorem	proof depth	running time	number of inference	number of solution	proof depth	running time	number of inference	number of solution
ls65	11	200.52	2944	281	11	489.05	2927	273
ls68	7	5.12	121	16	5	6.48	114	14
ls75	9	28.17	561	51	9	50.75	545	13
ls76t1	7	5.88	140	17	7	9.62	140	17
mqw	7	0.60	5	5	7	0.68	6	4
num1	9	0.80	14	6	9	1.10	14	6
prim	11	2.03	53	8	11	2.78	53	8
qw	9	0.77	10	4	9	0.85	10	4
rob1	7	0.35	2	2	7	0.35	2	2
rob2	11	5.88	299	4	11	7.90	298	2
schubert	32	65.30	1124	67	32	178.43	1124	68
shortburst	9	1.48	26	5	9	2.15	26	2
wos1	11	78.40	1154	90	9	201.72	1152	145
wos10	11	258.58	3981	233	11	1091.10	4143	115
wos11	11	281.20	4248	288	11	791.20	4336	130
wos12	7	0.73	27	3	7	0.97	27	3
wos13	7	8.43	301	51	7	11.17	301	51
wos14	9	10.87	313	49	9	13.57	313	48
wos15	14	8966.92	20553	2302	11	179.25	1594	102
wos17	9	47.52	1124	120	9	108.45	1124	12
wos19	9	89.05	1301	203	9	204.98	1291	194
wos2	9	5.00	166	8	7	7.00	159	8
wos23	7	1.85	48	2	7	2.77	48	2
wos24	9	19.45	443	60	9	29.23	419	56
wos25	9	57.22	828	167	9	75.77	793	129
wos27	9	28.22	542	94	7	51.97	522	92
wos29	9	106.13	1135	248	9	172.00	1135	235
wos3	7	0.63	14	3	7	0.85	14	2
wos30	7	1.62	40	4	5	1.08	19	3
wos31	18	5111.42	21510	552	18	6832.45	12120	250
wos32	5	1.73	21	7	7	8.13	67	6
wos33	9	7.85	90	7	9	12.53	86	6
wos4	11	653.38	7235	4	11	1988.20	7236	4
wos5	9	5.22	153	20	7	5.50	140	20
wos6	9	18.35	459	73	9	22.52	458	16
wos7	9	10.38	389	22	9	16.35	389	3
wos8	9	11.17	325	49	7	11.35	314	49
wos9	9	14.80	527	33	9	27.42	528	14

[3]The data are obtained on a SUN3/60 workstation with 12Mb memory. The Prolog system is the ALS Prolog Compiler (Version 0.60) from Applied Logic Systems, Inc.