Bildverstehen

Leonie Dreschler-Fischer 1
"Das 'Bootstrap-Problem' bei der geometrischen Szenenrekonstruktion - eine Übersicht"

Gerda Stein 16
"Konfliktlösung auf statistischer Basis bei der Bildanalyse mit Produktionsregeln"

Ingrid Walter 21
"EPEX: Bildfolgendeutung auf Episodenebene"

Michael Mohnhaupt 31
"On Modelling Events with an 'Analogical' Representation"

Natürlichsprachliche Systeme
und
die Verarbeitung gesprochener Sprache

Christa Hauenschild 41
"KI-Methoden in der Maschinellen Übersetzung?"

Jochen Dörre, Stephan Momma 54
"Generierung aus f-Strukturen als strukturgesteuerte Ableitung"

Martin Emele 64
"FREGE - Ein objektorientierter FRont-End-GEnerator"

Dekai Wu 74
"Concretion Inferences in Natural Language Understanding"

Hans Haugeneder, Manfred Gehrke 84
"Modelling Heuristic Parsing Strategies"

Peter Bosch (lag zum Druck nicht vor)
"Deeper Reasons for Shallow Processing"

Massimo Poesio 94
"An Organization of Lexical Knowledge for Generation"

R. Gemello, E. Giachin, C. Rullent 104
"A Knowledge-Based Framework for Effective Probabilistic Control Strategies in Signal Understanding"

Wissensrepräsentation und KI-Programmierung

Bernhard Nebel, Kai von Luck 114
"Issues of Integration and Balancing in Hybrid Knowledge Representation Systems"

Klaus Groth 124
"Der Aspekt der Zeitstruktur in zeitlogischen Formalisierungen der KI"

Stefan Wrobel 129
"Higher-order Concepts in a Tractable Knowledge Representation"

Helmut Simonis, Mehmet Dincbas 139
"Using Logic Programming for Fault Diagnosis in Digital Circuits"

Expertensysteme

Michael Beetz 149
"Specifying Meta-Level Architectures for Rule-Based Systems"

Werner Konrad, Andreas Jaeschke, Helmut Orth, Ono Tjandra 160
"Guiding the Maintenance of a Model-Based Advisory System by Explanation-based Learning"

Deduktive Systeme

Hartmut Freitag, Michael Reinfrank 170
"An Efficient Interpreter for a Rule-Based Non-Monotonic Deduction System"

Frank Puppe 175
"Belief Revision in Diagnosis"

Oskar Dressler 185
"Erweiterungen des Basic ATMS"

Ulrich Furbach 195
"Oldy but Goody - Paramodulation Revisited"

Marita Heisel, Wolfgang Reif, Werner Stephan 201
"Program Verification by Symbolic Execution and Induction"

R. Ulf Schmerl 211
"Resolution on Formula Trees"

Rainer Manthey, Francois Bry 221
"A Hyperresolution-based Proof Procedure and its Implementation in PROLOG"

Alexander Herold 231
"Narrowing Techniques Applied to Idempotent Unification"

Jürgen Müller 241
"THEOPOGLES - A Theorem Prover Based on First-order Polynomials and a Special Knuth-Bendix Procedure"

Selbstrepräsentierende Systeme

Pattie Maes 251
"Computational Reflection"

Kognitives Modellieren

Ab de Haan 266
"Cognitive Modelling and Education"

Spezielle Sektionen

Kognition - Wissensstrukturen beim Aufgabenlösen

Christian Freksa **277**

Barbara Becker 278
"Wissen und Können: Anmerkungen zur Wissensrepräsentation beim Aufgabenlösen"

Gerhard Strube 287
"Repräsentationsformen beim menschlichen Problemlösen"

Klaus Rehkämper 296
"Mentale Bilder und Wegbedeutungen"

Generierung in natürlichsprachlichen Systemen

Wolfgang Hoeppner **306**

Dietmar Rösner 307
"Generation of Content vs. Generation of Form: A Review of Recent Work in the SEMSYN Project"

Norbert Reithinger 315
"Ein erster Blick auf POPEL: Wie wird was gesagt?"

Helmut Horacek 320
"How to say WHAT - IT or SOMETHING"

Elisabeth Andre, Thomas Rist, Gerd Herzog 330
"Generierung natürlichsprachlicher Äußerungen zur simultanen Beschreibung von zeitveränderlichen Szenen"

Repräsentationssysteme für Grammatik und Lexikon

Günther Görz **339**

Dietrich Paulus 340
"Endliche Automaten zur Verbflexion und ein spezielles deutsches Verblexikon"

Ulrich Heid 345
"Zur lexikalischen Wissensquelle des Generierungssystems SEMSYN"

Stephan Busemann 355
"Generierung mit GPSG"

**Künstliche Intelligenz und Datenbanksysteme –
Systemarchitektur und konzeptuelle Modellierung**

Rudi Studer **365**

Gio Wiederhold, Surajit Chaudhuri, Waqar Hasan, Michael G. Walker,
Marianne Winslett 366
"Architectural Concepts for Large Knowledge Bases"

Heinrich Jasper 386
"Interfacing PROLOG and External Data Management Systems: A Model"

Theo Härder, Nelson Mattos, Bernhard Mitschang 396
"Abbildung von Frames auf neuere Datenmodelle"

A HYPERRESOLUTION-BASED PROOF PROCEDURE AND ITS IMPLEMENTATION IN PROLOG

Rainer Manthey and Francois Bry
ECRC
Arabellastr. 17
D-8000 Muenchen 81

1. Introduction

Our work on automated deduction has been motivated by database problems. The set of deduction rules and integrity constraints of a logic database can be considered as axioms of a first-order theory while the actual sets of facts constitute (finite) models of this theory. Satisfiability of the underlying axioms is a necessary prerequisite for any logic database. The procedure described in this paper is the basis of a program called SATCHMO (SATisfiability CHecking by MOdel generation) that has been implemented at ECRC as part of a prototype schema design system for logic databases.

Although SATCHMO has initially not been intended as a proof procedure, it turned out that a considerable amount of examples discussed in the theorem proving literature could be solved by model generation with remarkable efficiency. We have successfully tested our approach, e.g., on most of the 75 problems in [PEL86] as well as on a large collection of problems that we received from Argonne National Laboratory. Besides many encouraging results - Schubert's Steamroller has been solved in little more than a second, e.g. - we know about examples that lead to unacceptable results and that will probably never be solved by "pure" model generation.

SATCHMO's approach is based on hyperresolution. This inference rule allows to make benefit of the range-restrictedness of clauses. Range-restrictedness - introduced in [NIC82] - is a necessary condition for an efficient evaluation of queries against a database. Every variable in a range-restricted clause occurs at least once inside a negative literal in that clause. This property can be naturally expected for problems with a structured domain.

If applied to range-restricted clauses, hyperresolution always produces clauses that are ground. This allows to simulate exhaustive application of hyperresolution by means of unit hyperresolution and clause splitting. Unit hyperresolution fits particularly well to PROLOG. In addition, a depth-first implementation of clause splitting is well supported by PROLOG backtracking. The resulting PROLOG program for automatic model generation is stunningly short and simple. As nowadays PROLOG interpreters are offered for a huge variety of computers, theorem proving technology might become available to a wider audience if it could be based on standard PROLOG features instead of requiring special machines and languages.

The paper consists of six sections. After this introduction we give some basic definitions

around hyperresolution and introduce notations. In section 3 we elaborate on range-restricted clauses and show that every set of clauses can be transformed into a range-restricted set while preserving satisfiability. Model trees are introduced in section 4 and the correspondence between model tree search and hyperresolution saturation is shown. A PROLOG implementation of model tree generation is described in section 5 and its application to the Steamroller problem is discussed. In section 6 we give an improved procedure that is a decision procedure for a syntactically definable class of problems. Section 7 contains concluding remarks and hints to possible extensions. We don't give proofs within this paper.

2. Basic definitions and notation

Throughout the paper, Boolean connectives and/or/not/implies, resp., will be denoted by ,/;/~/--->, resp. Clauses will be represented in implicational form

$$A_1,...,A_m \text{ ---> } C_1;...;C_n$$

where $\tilde{}A_1$ to $\tilde{}A_m$ are the negative, C_1 to C_n the positive literals in the clause. Completely positive clauses are written as true ---> $C_1;...;C_n$, while completely negative clauses are implicationally represented in the form $A_1,...,A_m$ ---> false. Thus negation never occurs explicitly. We call the left-hand side of an implication *antecedent* and the right-hand side *consequent*. Clauses with antecedent 'true' will be called *statements*, all other clauses will be called *rules*.

The hyperresolution inference principle allows to derive a new statement - the *hyperresolvent* - from a single rule - the *nucleus* - and as many other statements - the *satellites* - as there are literals in the antecedent of the nucleus. Each of the antecedent literals has to be unifiable with a literal in one of the satellites. The respective unifiers must be compatible, i.e., a most-general unifier has to exist that allows to unify antecedent and satellite literals simultaneously. This mgu is applied to nucleus and satellites before the hyperresolvent is constructed by disjunctively conjoining the consequent of the nucleus with those satellite literals that do not occur in the antecedent of the nucleus. If all satellites consist of a single literal we speak of *unit hyperresolution*.

For any set P of statements and any set N of rules, hyp(N,P) denotes the set of all hyperresolvents that can be derived from a nucleus in N and satellites in P.

Let S denote a finite set of clauses, $S^+(S^-)$ the subset of statements(rules) in S. The *hyperresolution levels* of S can be inductively defined as $Hyp^0(S)=S^+$ and $Hyp^i(S)=Hyp^{i-1}(S)$ *Union* $hyp(S^-,Hyp^{(i-1}(S))$ for i>0. The *hyperresolution saturation* of S is the union over all hyperresolution levels and is denoted by Hyp(S).

In [ROB65] hyperresolution has been shown to be sound and complete for refutation, i.e., S is unsatisfiable iff Hyp(S) contains 'false'. In case of satisfiability, models of S can be extracted from Hyp(S) rather straightforwardly. For this purpose the ground instances of Hyp(S) over the ground terms in S (if any - a single artificial' constant else) have to be considered. Every subset M of the Herbrand base of S that covers each of these instances

induces a model of S in which exactly the atoms in M are true and every other atom is false. A set of ground literals is a *cover* of a set of ground clauses iff each of the clauses is subsumed by one of the literals. We call a model that is induced by a cover of the instance set of Hyp(S) a *h-model* of S.

3. Hyperresolution for range-restricted clauses

Many satisfiability problems are dealing with a non-uniform domain of interpretation, i.e., variables range over well-distinguished subdomains. This is in particular the case if problems are inherently many-sorted. Range-restrictedness requires that for every variable in a clause the subset of the domain over which the variable ranges is explicitly specified inside the clause. A clause is *range-restricted* if every variable in the consequent of the clause occurs in its antecedent as well. Range-restricted statements are necessarily ground. The class of range-restricted formulas consists mainly of those first-order formulas that can be equivalently expressed by means of restricted quantification.

For problems dealing with a single unstructured domain, range-restrictedness of clauses cannot be naturally expected. Examples of this kind can be found mainly among algebraic or set-theoretic problems. Especially non-ground statements have to be expected in this case. If a set S contains clauses that are not range-restricted, it nevertheless may be transformed into a set S^* that is range-restricted and that is satisfiable iff S is satisfiable. For this purpose an auxiliary predicate 'dom' has to be introduced and the following transformations and additions to be performed:

- Every statement true ---> C containing variables X_1 to X_n is transformed into a rule $dom(X_1),...,dom(X_n)$ ---> C.
- Every rule A ---> C such that C contains variables X_1 to X_n that don't occur in A is transformed into a rule $A,dom(X_1),...,dom(X_n)$ ---> C.
- For every ground term t that occurs in S, the unit clause true ---> dom(t) is added to S. If S is free of ground terms, a single unit true ---> dom(a) is added where 'a' is an artificial constant.
- For every n-ary predicate p and Skolem term t occurring as the i-th parameter of a p-atom in the consequent of a clause in S, a rule $p(X_1,..,X_i,..,X_n)$ ---> $dom(X_i)$ is added to S.

The predicate 'dom' makes explicit the domain of interpretation. The 'dom' literals added to non-range-restricted clauses provide an implicit instantiation of the respective variables over the whole domain. The additional clauses are necessary for guaranteeing that the relation 'dom' contains an entry for every ground term that occurs in Hyp(S). Although the transformation is not preserving equivalence in the strict sense, a kind of weak equivalence between S and S^* exists: if the relation assigned to the new auxiliary predicate 'dom' is removed from any model of S^*, a model of S is obtained. There is a one-to-one correspondence between models of S and models of S^* up to the 'dom' relation. Therefore the

transformation described preserves satisfiability as well.

Every set of clauses may be transformed into a range-restricted set in this way, but it has to be mentioned that the transformation may have the same effect as a partial instantiation.

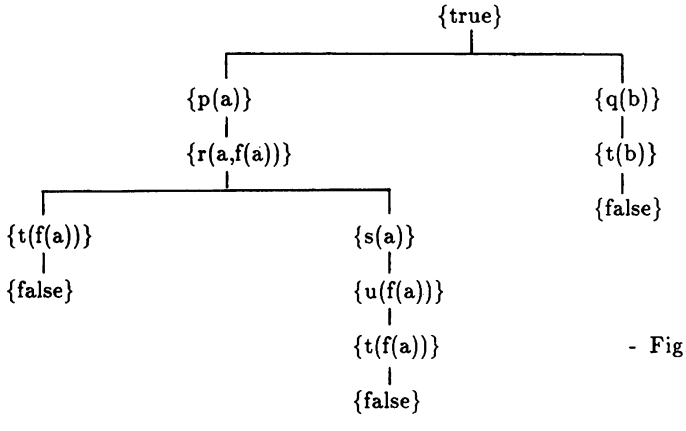
One can easily prove that Hyp(S) consists of ground clauses only iff all clauses in S are range-restricted. Hyperresolution saturation can be implemented very efficiently in this case. We will show that a combination of unit hyperresolution and clause splitting is sufficient in place of full hyperresolution. In addition, h-models can be determined particularly easily, as a ground Hyp(S) coincides with its instance set. The method we describe may therefore be interpreted as an implementation of a systematic search for covers of Hyp(S).

4. Model trees

In this section we investigate how to make benefit of the particularities of hyperresolution in the range-restricted case. As an introductory example let us consider the following range-restricted set S_1:

true ---> p(a) ; q(b)	r(X,Y) , u(Y) ---> t(Y)
p(X) ---> r(X,f(X))	q(X) ---> t(X)
r(X,Y) ---> t(Y) ; s(X)	p(X) , t(f(X)) ---> false
s(X) ---> u(f(X))	q(X) , t(X) ---> false

Instead of starting hyperresolution from p(a) ; q(b), one can analyse the case where p(a) is true independently from the case where q(b) is true. Such a splitting into two independent subproblems is sound because the clause p(a) ; q(b) is a ground disjunction. In each subcase unit hyperresolution is now applicable. Whenever non-unit hyperresolvents are obtained they are immediately split as well, i.e., new sub-subproblems are created. For each of the subproblems we finally will be able to derive 'false' by means of unit hyperresolution, which indicates that S_1 is unsatisfiable. Fig. 1 illustrates the different splitting and unit hyperresolution steps in form of a tree.



- Fig. 1 -

If the rule p(X),t(f(X)) ---> false would be missing in S_1, the two leftmost branches in this tree could be cut one step earlier - indicating the existence of two different h-models. The respective models - {p(a), r(a,f(a)), s(a), u(f(a)), t(f(a))} and {p(a), r(a,f(a)), t(f(a))} - can be directly extracted from the tree by collecting all literals along a branch not ending with 'false'. Because of this property we would like to call such a tree a *model tree*. Every tree T_S that satisfies the following conditions is a model tree for S:

1. The nodes of T_S are finite sets consisting of truth values or of ground literals from the Herbrand base of S.
2. The root of T_S is the set {true}.
3. Let N be any node of T_S and let anc(N) denote the union of N and all its ancestor nodes.

    a. N is a leaf iff either N contains 'false', or hyp(S^-,anc(N)) is empty.

    b. N has a direct descendant N' iff N' is a non-empty cover of hyp(S^-,anc(N))

We call leaves containing 'false' *failure nodes* and branches ending in a failure node *closed*. A leaf that does not contain 'false' is a *model node*. A model tree is closed if all its branches are closed.

Two important properties of model trees can be easily proved:

(*) If any model tree for S contains a model node N, then S is satisfiable with h-model anc(N).
(**) If any closed model tree for S exists, then S is unsatisfiable.

Thus model tree generation provides the basis for a sound and complete proof procedure for the class of clause sets with a hyperresolution saturation that is ground. As soon as a model node has been found, satisfiability has been proved due to (*). In case of unsatisfiability, however, the whole model tree has to be generated for being sure that all its branches are closed (**).

There are two cases in which satisfiability can be reported immediately: In case S consists of rules only, hyperresolution is not applicable and S is trivially satisfiable with empty h-model. If on the other hand no rule in S has the consequent 'false' no branch will ever be closed.

Model trees are related to special analytic tableaux in the sense of [Smu68]. The strategy by which this kind of tableaux can be constructed is such that a branch is expanded by splitting a clause C iff all negative literals in C are simultaneously unifiable with positive literals along the branch. This particular strategy could be expressed in terms of hyperresolution as well. Removing all negative literals from such a tableau leads directly to the corresponding model tree.

In general there are many different ways how a model tree may be constructed. Choices have to be made in which order to try the different possible covers of a partial saturation hyp(S⁻,anc(N)). In the next section we will present an implementation of a pure depth-first strategy for model tree generation. In general, Hyp(S) and therefore any model tree for S can be infinite. In case Hyp(S) is finite, however, all model trees for S are finite as well and model tree generation will always terminate. Therefore satisfiability is decidable for such sets of clauses. In section 6 we will further characterize this decidable class and give a more efficient procedure for deciding satisfiability of clause sets from this class.

5. A PROLOG-implementation of model tree generation

Unit hyperresolution can be implemented in PROLOG extremely easily and efficiently. After having defined '--->' as a PROLOG binary operator, problem clauses can be directly asserted as PROLOG facts. Solving the PROLOG goal '(A ---> C), A, not C' implements the search for a hyperresolvent C that is derivable with all satellites being units in the current database and C being "new", i.e., not subsumed by any of these units. (Note that all non-unit statements will be "derived" this way as well, because 'true' is a PROLOG goal that always succeeds). PROLOG's search through the '--->' -relation corresponds to the search for a nucleus of the next derivation step. Regarding antecedent and consequent of a clause as PROLOG goals (with Boolean connectives represented like in PROLOG) permits to implement the search for suitable satellites as well as the subsumption test via PROLOG goal interpretation. Unit hyperresolution saturation relative to a fixed set of satellites - as required in the definition of a model tree - can be achieved by means of a 'setof' procedure which is available as a system predicate in most PROLOG interpreters.

The following three predicates determine all covers of a given set of hyperresolvents - represented as a PROLOG list - successively on backtracking. The elements of the cover are asserted as facts into the PROLOG database and are automatically retracted on backtracking:

```
cover([ ]).
cover([H|T]) :-
    H, !, cover(T).
cover([H|T]) :-
    component(L, H),
    assume(L),
    cover(T).
```

```
component(L, (L; _)).
component(L, (_; D)) :- !, component(L, D).
component(L, L).

assume(X) :- assert(X).
assume(X) :- retract(X), !, fail.
```

Immediately after a cover has been computed we can already determine whether any of the descendants of this cover in the model tree under construction will contain 'false'. For this purpose we have made the choice not to treat completely negative clauses in the same way as the remaining clauses, but to represent every rule A ---> false as a PROLOG derivation rule 'false :- A'. This allows to cut branches of a model tree that lead to contradictions already one level before 'false' would be explicitly derived. If the PROLOG goal 'false' succeeds after the computation of any cover, backtracking is initiated and a different

choice for reaching a cover is made.

The whole proof procedure can now be programmed in form of two simple mutually recursive predicates:

```
satisfiable :-
    setof(C, ((A ---> C), A, not C), S),
    satisfiable(S).

satisfiable([ ]) :- !, not false.
satisfiable(S) :-
    cover(S),
    not false,
    satisfiable.
```

If 'satisfiable' succeeds, the database contains a h-model of the set of clauses in the 'rule' relation. Failure of 'satisfiable' indicates that no h-model can be found.

Despite of its simplicity this program appears to be surprisingly efficient if applied to problems that are naturally range-restricted. Schubert's Steamroller - introduced in [WAL84] - is an excellent example of this kind of problems. Our PROLOG representation of it corresponds to the 'standard' unsorted formulation of the problem given in [STI86]:

```
wolf(X) ---> animal(X)                    true ---> wolf(w)
fox(X) ---> animal(X)                     true ---> fox(f)
bird(X) ---> animal(X)                    true ---> bird(b)
snail(X) ---> animal(X)                   true ---> snail(s)
caterpillar(X) ---> animal(X)             true ---> caterpillar(c)
grain(X) ---> plant(X)                    true ---> grain(g)
fox(X),wolf(Y) ---> smaller(X,Y)          snail(X) ---> plant(i(X))
bird(X),fox(Y)--> smaller(X,Y)            snail(X) ---> likes(X,i(X))
snail(X),bird(Y) ---> smaller(X,Y)        caterpillar(X) ---> plant(h(X))
caterpillar(X),bird(Y) ---> smaller(X,Y)  caterpillar(X) ---> likes(X,h(X))
bird(X),caterpillar(Y) ---> likes(X,Y)
animal(X),animal(Y),smaller(Y,X),plant(W),likes(Y,W),plant(Z) --->
                    likes(X,Y) ; likes(X,Z).

false :- likes(X,Y), wolf(X), fox(Y).     false :- likes(X,Y), wolf(X), grain(Y).
false :- likes(X,Y), bird(X), snail(Y).
false :- animal(X), animal(Y), likes(X,Y), grain(Z), likes(Y,Z).
```

The best cputime Stickel has reported for the solution of this problem is 6 secs. Meanwhile [ENG87] has achieved 1 sec. We have applied the above program to the given formulation of the Steamroller (using a simplified 'setof' procedure for sets of ground elements) in interpreted C-Prolog (Vers. 1.5) on a VAX 11/785. Using the built-in predicate 'cputime' we have measured 1.35 secs. The model tree constructed by the program consists of 14 nodes and has 10 closed branches. If the theorem is omitted, we reach a h-model of 25 literals after 0.83 secs.

On the other hand there are cases where the implicit instantiation introduced through the transformation into range-restricted form makes model tree search impracticable because of the immense size of the tree. Problems 66 to 69 in [PEL86] - difficult function-problems suggested by Ch. Morgan - belong to this kind of examples.

6. An improved procedure for a syntactically defined class

As mentioned before model tree generation is a decision procedure for satisfiability for clause sets with a finite hyperresolution saturation. If we would know in advance that a given set belongs to this class we could use an even simpler (and in general more efficient) PROLOG implementation of model tree generation:

```
satisfiable_fin :-
        (A ---> C), A, not C, !,
              component(L,C),
              assume(L),
              not false,
              satisfiable_fin.
satisfiable_fin.
```

This predicate does no longer determine sets of hyperresolvents before trying to cover them. Instead it generates hyperresolvents as they come and covers them immediately. If the set of all possible hyperresolvents is finite, the program is guaranteed to exhaust this set and thus to reach 'false' whenever possible. In case Hyp(S) is infinite, 'satisfiable_fin' may work correctly as well, but there are cases where refutation-completeness is lost. The following clause set S_2 is an example for this:

```
true ---> p(a)                p(Y) ---> p(f(Y))
p(X) , q(X) ---> false        p(Z) ---> q(f(Z))
```

S_2 has infinitely many hyperresolution levels. The nodes of the model tree for S_2 are the sets {p(a)}, {p(f(a)),q(f(a))} and {p(f(f(a))),q(f(f(a))),false}, respectively. Thus model tree generation with the predicate 'satisfiable' will terminate as opposed to 'satisfiable_fin': this procedure reaches only the p-atoms in each level but never derives 'false'. Every atom is asserted immediately after its generation and thus leads to a new application of the same rule. The search function of 'satisfiable_fin' is simple, but inherently "unfair". Thus its simplicity can be exploited only for refuting sets with finite hyperresolution saturation.

The infinite generation of hyperresolvents is due to the presence of a special recursive rule in S_2: p(Y) ---> p(f(Y)) is recursive and in addition leads to the generation of infinitely many nested terms via hyperresolution. The Steamroller contains a recursive rule as well, namely

```
animal(X),animal(Y),smaller(Y,X),plant(W),likes(Y,W),plant(Z) --->
                              (likes(X,Y);likes(X,Z))
```

However, no new nested terms can be generated through this rule, as no Skolem term is involved in the recursion.

There is another phenomenon that influences a potential infinite generation of hyperresolvents. Consider the following set S_3:

```
true ---> p(a)                p(X) , s(Y,Z) ---> r(Z,f(X))
r(X,f(X)) ---> false          r(X,Y) ---> s(Y,f(X))
```

This set contains a cycle of recursion between the two rules on the right column that in-

volves Skolem terms as well. Despite this similarity with S_2, $Hyp(S_3)$ is finite and both procedures, 'satisfiable' as well as 'satisfiable_fin' terminate. This happens because the respective rules do not participate in any hyperresolution step.

A syntactical characterization of clause sets that contain a cycle of recursion leading to infinite term-generation when entered during hyperresolution can be given in terms of the connection graph of the set. Certain cycles in this graph can be distinguished by means of compatible unifiers, recursive substitutions and a reachability relation between atoms. The presence of these special cycles characterizes a solvable class of clause sets. This class properly contains two well-known solvable classes, namely the Bernays-Schoenfinkel class (clauses without non-constant Skolem terms) and the class of compact clause sets [LEW75] (sets without recursive rules). The formal definition of this class and a proof of its solvability will be given in a forthcoming paper.

7. Conclusion

In this paper a proof procedure for sets of range-restricted clauses has been proposed which is based on model generation via hyperresolution. The method exploits that hyperresolvents derivable from range-restricted clauses are ground. The procedure can be interpreted as generating a model tree in a depth-first manner. A PROLOG implementation of the approach has been described that is simple but allows to solve many problems with considerable efficiency. As a satisfiability preserving transformation into a range-restricted problem is always possible, the procedure is general purpose.

Two possible extensions are under investigation at the moment. First, one can use PROLOG derivation rules for representing other Horn clauses than only completely negative ones. Moreover, a careful use of non-ground PROLOG facts for representation of unit clauses can help to avoid many of those cases where a combinatorial explosion of generated facts due to implicit instantiation via 'dom' would otherwise occur. As long as it is guaranteed that non-Horn statements are always grounded before splitting, these deviations from a purely generative approach are acceptable and often very useful. However, it should be noted that PROLOG-specific problems due to recursion or missing occurs check may arise, that do not exist for the method described in this paper.

Second, we are going to make the method sound and complete for finite satisfiability as well. Most theorem provers do not terminate in many cases where a finite model exists. For applying the procedure in a database context, existence of finite models has to be detected. The price to be paid for being able to terminate more often in case of satisfiability will be a decrease in efficiency for refutation, however. In [BM86] we have investigated which additional features are required for achieving completeness for both, unsatisfiability as well as finite satisfiability. An extension of our method with these additional features appears to be straightforward.

Apart from the two points mentioned, further increase in efficiency may be obtained by means of more sophisticated strategies for model tree generation.

Acknowledgement:
We would like to thank L. Wos, R. Overbeek and the other members of the Argonne team for their hospitality and the stimulating discussions during the visit of one of the authors at Argonne. Furthermore we would like to acknowledge the many useful remarks made by the anonymous referees.

References:

[BM86] F. Bry and R. Manthey, *Proving finite satisfiability of first-order theories*, Internal Report KB-27, ECRC, 1986

[ENG87] D. Engelhardt, *Model elimination - Grundlagen und Implementation*, Diplomarbeit, Tech. Hochschule Darmstadt, Fachbereich Informatik, 1987

[LEW75] H. Lewis, *Cycles of unifiability and decidability by resolution*, Techn. Report, Aiken Comp. Lab., Harvard Univ., 1975

[NIC82] J.M. Nicolas, *Logic for improving integrity checking in relational databases*, Acta Informatica 18, 1982, 227-253

[PEL86] F.J. Pelletier, *Seventy-five Problems for Testing Automatic Theorem provers*, J. of Autom. Reasoning 2, 1986, 191-216

[ROB65] J.A. Robinson, *Automated Deduction with Hyper-Resolution*, Intern. Journ. of Computer Math. 1, 1965, 227-234

[SMU68] R. Smullyan, *First-order logic*, 1968

[STI86] M. Stickel, *Schubert's steamroller problem: formulations and solutions* J. of Autom. Reasoning 2, 1986, 89-101

[WAL84] C. Walther, *A mechanical solution to Schubert's steamroller by many-sorted resolution*, Proc. AAAI 1984, Austin(Tex.), 330-334
(rev. version: Artificial Intelligence 26, 1985, 217-224)