Some Experiments on the Applicability of Folding Architecture Networks to Guide Theorem Proving

Stephan Schulz*
Automated Reasoning Group
Institut für Informatik
Technische Universität München
Germany

Andreas Küchler†
Department of Neural
Information Processing
Fakultät für Informatik
Universität Ulm
Germany

Christoph Goller*
Automated Reasoning Group
Institut für Informatik
Technische Universität München
Germany

Abstract

One of the major problems in theorem proving is control of the proof search. A promising approach is the application of machine learning techniques for the acquisition of search control knowledge by learning from successful proof searches. In this paper we briefly discuss this idea and existing machine learning techniques for this task. We suggest neural *folding architecture networks* together with supervised training algorithms as a very promising candidate for learning search control knowledge. This suggestion is based on two sets of experiments in which we applied folding architecture networks to term ordering problems and clause classification tasks resulting from the proof search of the equational theorem prover DISCOUNT.

1 INTRODUCTION

Currently, automated theorem provers (ATP systems) and similar deductive systems are being introduced into more and more application areas, e.g. deductive databases, verification problems, or computer algebra systems. However, the power of current ATP systems is often insufficient for fully automatic operation. Therefore, more powerful proof systems are highly desirable.

Deductive systems can usually be described by two components: an inference engine, and a search strategy, which controls the application of inference steps. Inference engines are, by now, very efficient. Theorem provers like SETHEO [LSBB92], Otter [McC94], SPASS [WGR96], and DISCOUNT [ADF95] perform the inferences actually necessary for most proofs in a split second. Consequently ATP systems prove most simple problems much faster than humans. However, on more difficult tasks, theorem provers often break down completely, while human experts manage to cope with the new problems. Probably the most important reason for this is the experts meta-knowledge about proofs. Humans are capable of learning good strategies for new problems from experiences with previous ones. Most ATP systems, on the other hand, use fixed, preprogrammed strategies. Such hard-coded strategies can avoid a lot of redundancy, but they lack the adaptability necessary to be successful in different domains with very different characteristics.

A promising solution to this problem is to use machine learning techniques for the acquisition of search control knowledge from past successes and failures. Learning algorithms for this task need to deal with structured, arbitrary sized input (terms, clauses, tableaux). Most traditional machine learning techniques, both neural and symbolic, are not well suited to this task. One common solution is the use of a fixed set of *features* for the description of terms [SE90, Fuc95]. However, the selection of these features introduces a very strong bias, and severely limits the class of relations that can be learned.

Some symbolic approaches work directly on structures. *Explanation Based Generalisation (EBG)* and related techniques (e.g. [MCK+89, KW94]) can speed up the proof search, but are restricted to learn shortcuts (*lemmata*) deducible within the calculus of the prover. EBG cannot capture heuristic knowledge, which is usually approximative and results from inductive inference, generalising multiple observed cases. *Learning by pattern memorisation* [DS96a] is restricted to learn information about finite subsets of the relevant structures, while *learning by term space mapping* [DS96a] is still in an infant stage.

Connectionist techniques have recently been generalised to deal with structured input. *Folding architecture networks (FA-Networks)* with gradient-based training algorithms like *back-propagation through structure (BPTS)* [GK96, KG96] may be viewed as generalisation of the well-known discrete-time recurrent neural network models [GKW94]. The generic architecture has been proven to be a universal approximator for mappings from directed acyclic graphs to real vector spaces [Ham96]. In this paper we will present experimental results that indicate that FA-Networks seem to be well suited for the acquisition of search control knowledge and for guiding the deductive process of many symbolic reasoning systems.

*Email: {schulz,goller}@informatik.tu-muenchen.de
†Email: kuechler@informatik.uni-ulm.de

The main part of this paper is structured as follows. First we briefly discuss the learning of heuristics for theorem provers and suggest two relevant subtasks. Next, we give a short description of FA-Networks. The main new contributions of this paper are the experimental results in Sec. 4. We conclude the paper with an outlook on future work.

2 LEARNING HEURISTICS FOR THEOREM PROVERS

ATP systems try to show that a conjecture (the *goal*) is a logical consequence of a set of axioms. *Bottom-Up* provers, like Otter and DISCOUNT, achieve this by *saturating* the set of input clauses (and possibly the negated theorem), i.e. they derive new facts until the theorem can be proved (either directly or by contradiction). The order in which new consequences are considered is the most critical factor for the power and efficiency of the prover. Top-down systems, e.g. SETHEO, enumerate possible decompositions (*tableaux*) of the negated goal. In this case the order in which possible tableaux are explored is critical for the performance.

An optimal heuristic would compute a total preference ordering on all possible inferences for each state during the proof search. This optimum, however, is very hard to realise. The description of a state in the proof process is extremely large, and the number of potential successor states usually immense. Consequently, the overhead of such an approach is unbearable. Most conventional heuristics only control a single high level choice-point (e.g. the selection of the next fact[1] to process), and learning heuristics in current provers mimic this behaviour. For saturation-based provers in particular, heuristics ignore most of the state of the proof process and concentrate only on the fact to be evaluated (and possibly the goal). In this case, a preference ordering can be established by assigning a numerical rating to each new fact exactly once.

It is of course impossible to exactly deduce which facts will contribute to the solution of a given problem. Currently there are only some ad-hoc ideas about the features and concepts useful in distinguishing probably useful facts from probably useless ones. To automatically learn good evaluation functions, the applied machine learning algorithm therefore has to be able to *discover* the necessary concepts by itself. We will evaluate the ability of FA-Networks and BPTS to inductively infer such concepts in the field of term orderings and then apply FA-Networks for the classification of facts from proof searches.

2.1 Learning Term Orderings

Term orderings have proven to be very useful in restricting and guiding search for theorem provers. *Ordered resolution*, the superposition calculus (used e.g. in SPASS), and completion-based techniques for equational deduction all use term orderings to improve proof search. Such orderings can therefore be considered to be a typical example of a non-trivial concept helpful in guiding automated deduction. We consider the ability to rediscover this concept from examples only to be a good indication of the ability of a machine learning algorithm to cope with problems in the field of automated deduction.

For this reason we analysed the behaviour of FA-Networks on two frequently implemented term orderings: The *Lexicographic Path Ordering (LPO)*, suggested by Kamin and Levi as a variant of the *Recursive Path Ordering* [Der79], and the *Knuth-Bendix-Ordering (KBO)*, suggested in [KB70].

The LPO uses a total precedence on function symbols, recursive descent, and lexicographic comparison among subterms, to compare composite terms. Due to the recursion (and possible backtracking), comparing two large terms with the LPO is rather complex.

The KBO calculates the sum of weights assigned to function symbols and variables to compare two terms, and uses a precedence on function symbols to break ties. As an additional constraint, a term can only be larger than another term if each variable occurs in it at least as often as in the potentially smaller one. Due to space restrictions, we report detailed results for the LPO only.

2.2 Classification of Facts

A simple and direct approach for learning heuristics is the classification of facts into those likely to lead to a proof and those that are probably useless. In most cases, the number of facts contributing to a proof is extremely small compared to the number of all facts generated (see [DS96b] for some examples). An average proof search may generate 100 000 facts, about 100 of which contribute to the proof. Training on such large sets of examples is very time consuming and not promising. It also turns out to be unnecessary.

Assuming a perfect fact selection, only direct derivatives of useful facts can ever be constructed during a proof search. It is therefore sufficient to learn evaluation functions that prefer positive examples to negative ones from this smaller set. The gain in efficiency does come at a price, of course, as small errors may open up regions in the search space that were not covered by the training examples. Our experiments will ignore this problem, as we are mainly interested in the basic ability of FA-Networks to classify these facts.

3 FOLDING ARCHITECTURE NETWORKS

Folding architecture networks are a recursive variant of the standard multi-layer perceptron (MLP). They work on finite labeled directed acyclic graphs (DAGs). This paper will deal with networks for the classification of terms, which can easily be represented as DAGs. These networks compute functions from the domain of (ground) terms into the interval $[-1, 1]$. For a more detailed discussion of

[1] We use the term *fact* to denote all pieces of knowledge handled by the inference engine, e.g. rules, clauses, and equations.

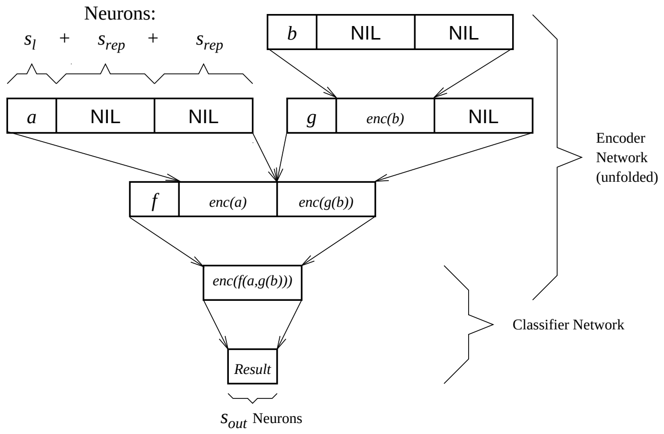

Figure 1: Network (virtually) unfolded to accommodate $f(a, g(b))$

the folding architecture and BPTS, and for experimental results on artificial problems, see [GK96, KG96].

Let F be a finite set of symbols, let $a : F \to \mathbb{N}$ describe the arity of these symbols and let $a_{max} := max\{a(f)|f \in F\}$ be the maximum arity of all symbols in F. $Term(F)$ denotes the set of (ground) terms over F. The function $label : F \to [-1, 1]^{s_l}$ assigns a label (a vector of s_l reals) to each function symbol. $s_{rep}, s_{out} \in \mathbb{N}$ are two constants, representing the size of the *compressed representation* of a term and of the network's output, respectively[2].

A FA-Network is described by two MLPs with sigmoidal activation functions, the recursively applied *encoder network* N_{enc}, and the *classifier network* N_{class}. The encoder network has $s_l + a_{max} \times s_{rep}$ input neurons and s_{rep} output neurons. Thus, N_{enc} computes a function $f_{enc} : [-1, 1]^{s_l + a_{max} \times s_{rep}} \to [-1, 1]^{s_{rep}}$. Let $\bullet$ denote the vector concatenation. Then the function $enc : Term(F) \to [-1, 1]^{s_{rep}}$ is defined recursively by
$enc(g(t_1, \ldots, t_{a(g)})) :=$
$f_{enc}(label(g) \bullet enc(t_1) \bullet \ldots \bullet enc(t_{a(g)}) \bullet \mathrm{NIL}^{(a_{max} - a(g))})$
We say the network is *virtually unfolded* to accommodate the term. Constants are encoded first, composite terms are encoded using the representation of their subterms. The NIL vector is a fixed representation for the absence of a subterm. Fig. 1 shows an example where both networks are single-layer perceptrons.

The classifier network, N_{class} has s_{rep} input neurons and s_{out} output neurons. Thus, it computes a function $f_{class} : [-1, 1]^{s_{rep}} \to [-1, 1]^{s_{out}}$. By using the output of the encoder as the input of the classifier, the total network defines $F_{net} := f_{class} \circ enc : Term(F) \to [-1, 1]^{s_{out}}$.

The network is trained by *back-propagation through structure*, a recursive variant of standard back-propagation. The error at the output layer of the classifier network is propagated back through the entire virtually unfolded network. Thus, the resulting term representations are directly optimised to the task at hand. Current refinements include an efficient representation of terms as minimal DAGs[3]. This speeds up learning, as the complexity of BPTS per epoch is linear in the total number of nodes and edges in the training set.

We call a network where both the encoder network and the classifier network consist of only input- and output layers a *simple FA-Network*. If we only consider signatures with $a_{max} = 1$, the corresponding simple FA-Networks are discrete-time recurrent networks [GKW94], and all known results for this class of models apply.

4 EXPERIMENTS

In this section we will describe the experimental setup and the results for a number of experiments on orderings and term classification. All examples have been produced by the DISCOUNT system for equational deduction in sequential proof runs on a set of 29 problems of medium difficulty. The terms and learning tasks can be considered to be quite typical for completion-based equational reasoning.

The problems have been taken from the domain of lattice ordered groups and share a common signature. They use 4 function symbols of arity one and two, a special constant (the neutral element of the group) and up to 4 Skolem constants (mostly appearing in the goals only). All terms are represented with normalised variables[4], which have been treated as additional constants for the label encoding. Labels were encoded with binary values (from $\{-1, 1\}$) using 5 positions: One to denote the status of the symbol (variable or operator), and 4 to guarantee uniqueness. Experiments with more sophisticated label encodings gave no significant differences in the results.

All reported results have been obtained by a 10-fold stratified cross-validation with 3 random weight initialisations per fold, i.e. each data point shown represents the mean value of 30 distinct experiments. Each training process was carried out in batch modus (weight update after the presentation of the whole training set) and stopped after 2000 learning epochs.

4.1 Experiments with the LPO

Comparisons with respect to a term ordering occur very frequently in a completion-based theorem prover. Thus, we were able to collect a data set of sufficient size (2921 term pairs) by only including the original term pairs for which the inference engine requested the comparison, and discarding comparisons resulting from recursive descent within the comparison routines. For experiments with the LPO, the data set was split into three different classes. Class 1 contained 604 unique equations were the left term

[2] The choice of $label$ and s_l is analogous to the encoding of features for a conventional MLP, while the choice of s_{rep} and s_{out} corresponds to decisions about the size of the hidden and output layers.

[3] Minimal DAGs represent a set of terms with a minimal number of nodes and edges by representing multiple occurrences of substructures only once.

[4] That is, variables within a term are always taken from the same set and appear in the same order.

is smaller w.r.t. the given ordering, class 2 contained 789 unique equations where the left term is larger, and class three consisted of 1528 unique equations with incomparable terms. We carried out three different experiments: Separation of class 3 from classes 1 and 2, separation of classes 1 and 2, and separation of all three classes in one experiment. The target outputs were chosen as -1, 1 and 0 for classes 1, 2 and 3, respectively. A term was counted as correctly classified if the absolute value of the difference between target and output of the network was less than 1 (less than 0.3 for 3-class experiments).

In all cases the network performed very well even with a small representation layer, and little overfitting could be detected. Table 1 shows selected results (after 500, 1000 and 2000 epochs) and standard deviations for both training (Tr. %) and test (Te. %) sets. It also includes the percentage of examples in the largest class of the test set – i.e. the percentage that a naive guesser would achieve by always predicting this class. The results for the KBO are very similar.

Ep.	$s_{rep} = 2$		$s_{rep} = 5$		$s_{rep} = 7$	
	Tr. % ± std.	Te. % ± std.	Tr. % ± std.	Te. % ± std.	Tr. % ± std.	Te. % ± std.
Separate class 3 from classes 1 and 2 (largest class: 51%)						
500	80.72	79.88	95.09	94.43	97.30	96.47
	1.90	3.03	1.42	2.40	0.59	1.36
1000	86.45	86.14	97.67	97.14	98.76	97.61
	2.26	3.29	0.82	1.28	0.42	1.27
2000	88.23	88.09	98.32	97.55	99.17	97.96
	2.89	3.15	0.73	1.29	0.44	1.00
Separate class 1 from class 2 (largest class: 55.7%)						
500	93.03	93.02	99.64	99.26	99.73	99.28
	5.63	5.41	0.20	0.63	0.16	0.62
1000	98.32	98.13	99.96	99.45	99.97	99.52
	1.70	1.57	0.11	0.66	0.06	0.54
2000	99.26	98.78	100.00	99.50	99.99	99.35
	0.62	0.95	0.01	0.56	0.06	0.63
Separate all three classes (largest class: 49%)						
500	65.22	64.92	95.56	95.02	94.40	93.96
	17.10	17.82	1.97	2.28	3.13	3.10
1000	91.96	91.68	98.42	97.45	98.27	97.54
	5.41	5.60	0.36	1.13	0.60	1.02
2000	96.20	95.99	98.96	97.76	98.96	97.96
	2.69	2.75	0.31	1.05	0.28	0.77

Table 1: Selected results from the LPO-Experiments

4.2 Experiments in Fact Classification

The data for these experiments was generated from PCL listings [DS96b] that contained all equations generated during the 29 proof searches. Equations immediately simplified or subsumed were discarded, as well as those generated, but never used for further inferences. From the remaining equations we selected those contributing to the proofs and those up to two *critical pair* inferences removed from them. This resulted in only 267 distinct equations (out of nearly one million equations generated during the 29 proof runs).

For our first experiment, all equations contributing to *any* proof were marked as positive examples, and all equations contributing to *no* proof served as negative examples. Each equation was present only once in the total data set. There were 113 positive examples and 154 negative examples in the set, so that a trivial guessing strategy could achieve 58%. The generalisation (around 70% for $s_{rep} = 5, 10$) was not as good as for the ordering experiments, and some overfitting problems became obvious. Nevertheless, even small improvements in the selection of facts can result in dramatic improvements in the overall performance of a prover. Table 2 gives an overview of results for different sizes for the compressed term representation, and Fig 2 shows the learning curves for $s_{rep} = 10$ in more detail.

Ep.	$s_{rep} = 1$		$s_{rep} = 5$		$s_{rep} = 10$	
	Tr. % ± std.	Te. % ± std.	Tr. % ± std.	Te. % ± std.	Tr. % ± std.	Te. % ± std.
Equations without goal (largest class: 58%)						
500	59.07	55.82	86.84	74.05	97.12	69.32
	4.20	9.47	3.50	7.90	1.50	8.17
1000	63.46	59.43	93.66	71.33	99.43	70.94
	4.65	9.36	2.28	7.84	0.52	7.75
2000	66.48	62.41	95.71	72.82	99.47	69.53
	5.15	9.39	1.57	7.37	0.53	6.70
Equations annotated with goal (largest class: 55%)						
500	57.76	57.23	80.27	70.35	88.37	70.10
	2.52	4.58	1.50	4.96	2.31	4.25
1000	63.14	61.06	83.93	70.50	93.39	69.90
	1.98	5.92	1.65	4.68	1.70	4.47
2000	64.31	62.15	86.02	70.35	95.11	68.76
	2.70	5.35	1.31	4.26	1.15	4.97

Table 2: Selected results from the fact classification experiments

For the second experiment in fact classification, we decided to give the neural network additional information, namely, to encode the (initial) goal to be proved with each clause. Each equation was presented as a term $peq(g, e)$, where g and e are equations representing goal and processed fact. This resulted in a much larger data set, as identical facts from different proof attempts became separable. The class label distribution of the original data set (1025 examples, split in 301 positive examples and 724 negative ones) was balanced by selecting each negative example with a probability of 0.5. This results in a data set of 671 examples, 301 positive and 370 negative ones (55% of the 671 examples). Consult table 2 for selected results. The addition of goal information seems to give no significant improvement of the generalisation performance in this experiment (compare Fig. 2 for $s_{rep} = 10$). The performance on the training set is worse than for the previous experiment (with smaller terms). This may indicate that the concepts involved are more complex and require more neurons and longer training times to be captured. However, the number of examples seems to be too small to train large networks, resulting in the overfitting phenomenon visible for the case $s_{rep} = 10$ and even more pronounced for $s_{rep} = 15$ (there are 260 free parameters (weights) in the net with $a_{max} = 2, s_{rep} = 10$ and $s_{out} = 1$, and 540 free parameters for the case $s_{rep} = 15$).

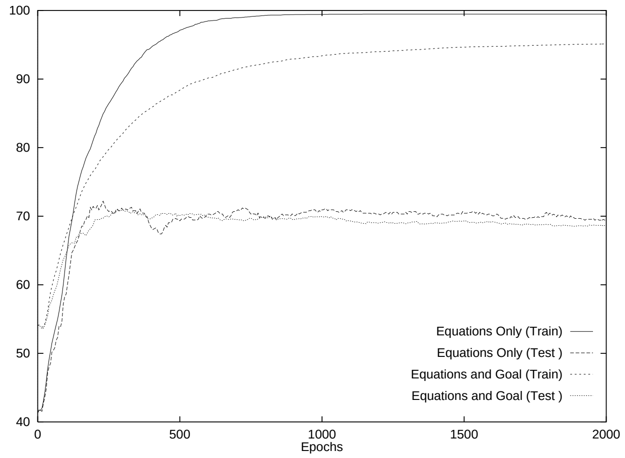

Figure 2: Learning curves for fact classification problems ($s_{rep} = 10$), the y-axis shows performance in %.

The smaller standard deviation is most likely a result of the greater size of the data set (671 examples compared to 267 for the former experiments).

5 CONCLUSION AND FUTURE WORK

As our experiments show, FA-Networks do have the power to learn standard term orderings, and are also able to predict the importance of a fact to a certain extend. Due to the (comparatively) small size of the training sets, it is not yet possible to conclude whether the inclusion of goal information leads to a real improvement for learning search heuristics. Nevertheless, FA-Networks seem to be suitable learning devices for the acquisition of search control knowledge for theorem provers and similar systems.

The next task is the application of FA-Networks in an existing theorem prover. The Automated Reasoning Group in Munich is currently integrating the SETHEO prover and FA-Networks. After a first test phase, transfer to the DISCOUNT system will also be considered.

Another important task is the improvement of FA-Networks. Currently, FA-Networks require the same signature for both training and application data. Pattern-based techniques or signature matches can be used to transfer learned knowledge to problems with different signatures. Another interesting research topic is the encoding of a-priori knowledge into the term representations.

Finally, to avoid long training times, a database of trained networks for specific application domains may be assembled, and criteria for the selection of a suitable net from this database need to be developed.

Acknowledgments

This research was partially supported by the German Research Foundation (DFG) under grant No. Pa 268/10-2 and by the Commission of the European Union (Esprit BRA 9119 MIX).

References

[ADF95] J. Avenhaus, J. Denzinger, and M. Fuchs. DISCOUNT: A System for Distributed Equational Deduction. In *Proc. of the 6^{th} RTA, Kaiserslautern*, number 914 in LNCS, pages 397–402. Springer, 1995.

[Der79] N. Dershowitz. Orderings for Term Rewriting Systems. In S.R. Kosaraju, editor, *Proc. of the 20^{th} Anual Symposium on Foundations of Computer Science, San Juan*, 1979.

[DS96a] J. Denzinger and S. Schulz. Learning Domain Knowledge to Improve Theorem Proving. In M.A. McRobbie and J.K. Slaney, editors, *Proc. of the 13^{th} CADE, New Brunswick*, number 1104 in LNAI, pages 62–76. Springer, 1996.

[DS96b] J. Denzinger and S. Schulz. Recording and Analysing Knowledge-Based Distributed Deduction Processes. *Journal of Symbolic Computation*, 21:523–541, 1996.

[Fuc95] M. Fuchs. Learning Proof Heuristics by Adapting Parameters. In *Proc. of the 12^{th} International Workshop on Machine Learning*, pages 235–243, San Mateo, CA, 1995. Morgan Kaufmann.

[GKW94] C.L. Giles, G.M. Kuhn and R.J. Williams, guest editors. Special Issue on Dynamic Recurrent Neural Networks. *IEEE Transactions on Neural Networks*, 5(2), 1994

[GK96] C. Goller and A. Küchler. Learning Task-Dependent Distributed Representations by Backpropagation Through Structure. In *Proc. of the ICNN-96*, volume 1, pages 347–352. IEEE, 1996.

[Ham96] B. Hammer. Universal Approximation of Mappings on Structured Objects using the Folding Architecture. Technical Report, Reihe P, Heft 183, Universität Osnabrück, Fachbereich Informatik, 1996.

[KB70] D.E. Knuth and P.B. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Algebra*, pages 263–297. Pergamon Press, 1970.

[KG96] A. Küchler and C. Goller. Inductive Learning in Symbolic Domains Using Structure-Driven Recurrent Neural Networks. In G. Görz and S. Hölldobler, editors, *Proc. of the 20th German Annual Conference on Artificial Intelligence, Dresden (KI-96)*, number 1137 in LNAI, pages 183–198. Springer, 1996.

[KW94] T. Kolbe and C. Walther. Reusing Proofs. In *Proc. of the 11^{th} European Conference on Artificial Intelligence*, pages 80–94, 1994.

[LSBB92] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Perormance Theorem Prover. *Journal of Automated Reasoning*, 1(8):183–212, 1992.

[McC94] W.W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, 1994.

[MCK+89] S. Minton, J.G. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzioni, and Y. Gil. Explanation-Based Learning: A Problem Solving Perspective. *Artificial Intelligence*, 40 (Special Volume: Machine Learning), 1989.

[SE90] C.B. Suttner and W. Ertel. Automatic Acquisition of Search Guiding Heuristics. In M.E. Stickel, editor, *Proc. of the 10^{th} CADE, Kaiserslautern*, number 449 in LNAI, pages 470–484. Springer, 1990.

[WGR96] C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER Version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *Proc. of the 13^{th} CADE, New Brunswick*, number 1104 in LNAI, pages 141–145. Springer, 1996.