IMPLEMENTING DYNAMIC SEMANTIC RESOLUTION

Simon Thornton [1]

[1] Dept. of Computer Science, The Australian National University

ABSTRACT: Automated theorem provers have many important practical applications. However, there are a number of obstacles to theorem provers becoming truly powerful tools. This paper gives a brief outline of the concepts behind a modern theorem prover. The implementation and testing of a rule of inference called dynamic semantic resolution is then discussed. The rule uses semantic imformation to guide a proof search. The results indicate that, while not without problems, dynamic semantic resolution has the potential to be a powerful rule of inference.

INTRODUCTION

Automated theorem provers are computer programs capable of artificial reasoning. Given a problem expressed in a special form, theorem provers are able to generate answers, usually by proof by contradiction. Theoretically, automated theorem provers are capable of solving any problem expressed in a formal logic. In practice, whilst still being capable of solving a great number of problems rapidly, today's automated theorem provers are not especially smart.

Despite this, theorem provers still find applications in the fields of mathematics and logic. They are also incredibly important in the design and verification of logic circuits and computer programs, helping to avoid costly errors such as the Pentium bug and to ensure the bug free execution of critical programs such as those that control railway signal switches.

Given their potential applications, there is much research into improving the speed and efficiency of automated theorem provers. [Wos88] describes a number of obstacles to automated theorem provers being truly powerful research tools. The algorithm for semantic resolution described in this paper is an attempt to overcome some of these problems, specifically those relating to search focus and the generation and retention of new information.

AN OUTLINE OF AUTOMATED THEOREM PROVING

In order to aid in understanding the rest of the paper this section gives a very general outline of some of the concepts associated with automated theorem proving. For a more in depth guide see [CL73].

For most modern theorem provers the basic unit of input is a clause. A clause is the disjunction of a set of literals. A literal can be thought of as a statement that is true or false. In first order logic a literal is equivalent to a predicate, in propositional logic it is an atomic proposition. It is the literals that are given truth valuations in a model. At any particular time a theorem prover will choose a clause to be resolved with an existing set of clauses. This clause is called the given clause.

To generate a proof from the input clauses the theorem prover uses a rule of inference. Such a rule allows new information, in the form of new clauses, to be generated from previous clauses. Dynamic semantic resolution is an example of a rule of inference, the development of which is discussed below.

Usually, automated theorem provers are attempting to find a proof by contradiction. To do this a potential theorem is negated and included in the input clauses. The theorem prover then uses its rule of inference to generate new clauses. If it is able to infer a contradiction, that is a clause of the form A and a clause of the form `not A`, then the proof of the theorem is complete.

THE BACKGROUND OF DYNAMIC SEMANTIC RESOLUTION

Most modern theorem provers use a purely syntactic method for generating proofs. In a syntactic method the meaning of the symbols is irrelevant, the computer doesn't understand what it is manipulating. The basic rule of inference that uses this method is binary resolution, first described by Robinson in [Rob65a]. The intuition underlying this rule can be thought of as the 'clashing' away of a literal that is negated in one existing clause and un-negated in another to produce a new clause. Figure 1 illustrates this process.

$$\frac{A \vee B \vee D \qquad C \vee -A \vee E}{B \vee D \vee C \vee E}$$

Figure 1: The existing clauses are shown above the line. Using binary resolution the literal A is 'clashed' away to produce the new clause below the line. Note: -A means not A or the negation of A.

Robinson later described an improved resolution algorithm, which he called hyper-resolution, in [Rob65b]. Hyper-resolution was able to deal with more than two clauses at a time, leading to bigger inference steps and faster proofs.

In [Sla67] Slagle showed that hyper-resolution was a special case of a more general rule known as semantic resolution. This rule uses a model, a truth valuation of the syntactic symbols, to guide the inference steps. When using semantic resolution the theorem prover is no longer blindly manipulating symbols as it does with binary resolution; instead it has some concept of their meaning. The special case of semantic resolution that Robinson called hyper-resolution uses a static, trivial model, where everything is false or everything is true, to guide the proof search.

By contrast, dynamic semantic resolution (DSR) uses a changing model to guide the theorem prover. The model is developed as the proof search progresses so that the guidance provided is closely adapted to the search's development. However, the dynamic and arbitrary nature of the model can cause problems. This paper discusses the implementation and testing of an algorithm developed by John Slaney for dynamic semantic resolution. The results of this work are presented, along with a brief evaluation of DSR as a rule of inference, both in terms of its ability to solve problems and its efficiency in comparison to existing rules of inference.

THE SEMANTIC RESOLUTION ALGORITHM

Slaney's algorithm for DSR builds a single semantic resolution inference from a number of semantically constrained binary resolution steps. The algorithm consists of three main sections; `STARTUP`, the `MAIN LOOP` and the `Saturate Kept` routine. This latter routine is used by both the `STARTUP` and `MAIN LOOP` sections of the algorithm.

Initially, the input clauses are all kept in their own set. During `STARTUP` the theorem prover chooses an input clause and resolves it with the `nuclei`. The chosen input clause is then modelled and moved to the set of `nuclei` if it is true or both the set of `nuclei` and the set of `satellites` if it false. Any clauses generated from the resolution are saturated against the `satellites` using the `Saturate Kept` routine. Those generated clauses which are false in the model are added to the `passive` set. Once the input clauses have all been chosen given clauses are taken from the `passive` set. This is done in the `MAIN LOOP` section of the algorithm.

A single iteration of the `MAIN LOOP` corresponds to a single semantic resolution inference. Each resolution within this iteration is a single binary resolution step. The recursive `Saturate Kept` routine uses these steps to build the semantic resolvents. The `nuclei` are the set of input clauses. The set of `satellites` consists of both those input clauses and those given clauses which are false in the model. The movement of clauses in the main loop of the algorithm is illustrated in figure 2 below.

Figure 2 shows how the `MAIN LOOP` uses a variation on the Set of Support algorithm described in [WRC65]. Essentially, the theorem prover has two active, or useable, sets; the `nuclei` and the `sattelites`.

A given clause taken from the `passive` set is first resolved with the `nuclei`. The given clause is then modelled and discarded if it is true in the model, or added to the set of `satellites` if it is false. Any

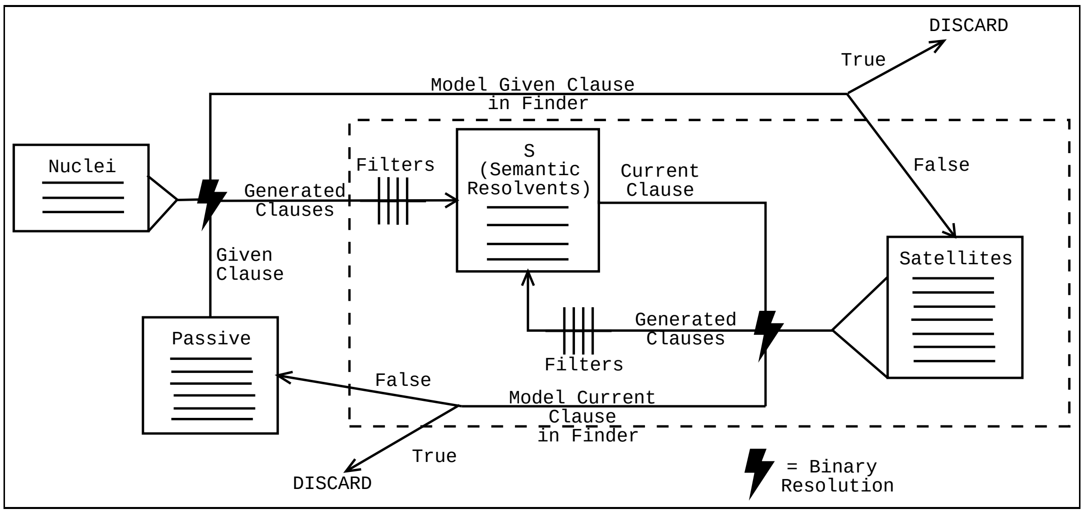

Figure 2: A graphical representation of the main loop of the dynamic semantic resolution algorithm.

clauses generated by the binary resolution with the nuclei are then added to the set S of resolvents and saturated against the `satellites` until no literals from the original clause remain. This rescursive saturation procedure is the `Saturate Kept` routine and is contained within the dashed box in figure 2.

After each resolution with the `satellites` any new clauses are added to the set of resolvents, provided they pass through the filters applied. The current clause from S is then added to the `passive` set if it is false in the model, or discarded if it is true. This saturation procedure continues until the set of resolvents S is empty.

Apart from the use of two active sets the DSR algorithm differs from more traditional rules of inference in the use of the model to filter out clauses. At every step clauses which are true in the model, and therefore less likely to generate a contradiction, are discarded. The other key point is the requirement that for generated clauses to be kept they must have resolved on an ‘original’ literal. The resolvents of each binary resolution will have literals from both parent clauses. Once the saturation procedure is commenced it is possible that the literal in the current clause chosen to be ‘clashed’ away comes from a satellite, not the given clause or the nuclei. This is not allowed in the algorithm and resolvents produced in this way are discarded by the filters.

IMPLEMENTATION DETAILS

The program that implements the above algorithm is called Sem_res. Initially written in C the program was translated to Java during the project in order to provide greater stability and flexibility. Sem_res links together two pre-existing programs: an automated theorem prover called Otter written by William McCune and a constraint solver called Finder written by John Slaney. The only changes that were made to the existing programs was the commenting out of the contents of a particular routine in Otter that performed ‘factor simplification’ on the generated clauses. This was done to prevent literals that may have been used during the saturation procedure from being deleted.

In order to generate a proof Sem_res runs Otter and Finder in turn, analysing the output from one in order to generate the input for the other. This flow of data is illustrated in figure 3 below. Hopefully, Otter will eventually discover a contradiction in its input and the theorem will be proved.

At each stage the input and output of Otter and Finder are treated as a number of strings. Each string is stored within a Clause object. These Clauses are stored within Sem_res in a number of linked lists, one for each of the nuclei, satellites and passive sets, as well as a number of temporary ones used during the saturation procedure. The Finder module also uses a linked list to store all of the clauses that are

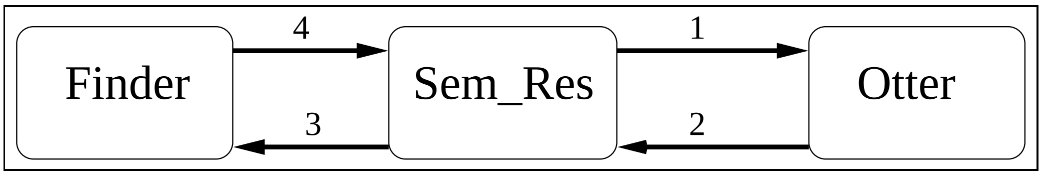

Figure 3: The flow of data in a single binary resolution inference step within the algorithm for semantic resolution.

in the current model. Using strings to store the clauses within Sem_res allows easy conversion of Otter syntax to Finder syntax for the purposes of modelling each clause.

The current version of Sem_res requires a number of input files for each problem that it is asked to solve. These files specify the options for Otter and the input constraints for Finder, as well as the input clauses themselves. The automatic generation of the constraints for Finder is an obvious improvement that could be made to the program.

RESULTS

Initial Results

The creation of Sem_res took place in two parts. The first incarnation only approximated Slaney's algorithm. The filters applied to the generated clauses were rough and the most important one, relating to the literals on which resolution was allowed, was not implemented at all. Despite this, the program was able to demonstrate an improvement over unguided binary resolution as the results in table 1 indicate.

Table 1: Initial results from DSR approximation

Problem	**Otter (Binary resolution)**		**Sem_res (DSR)**	
	Given Clauses	Generated Clauses	Given Clauses	Generated Clauses
Condensed Detachment 1	906	70656	22	205
Condensed Detachment 2	667	50115	60	970
Syntactic	1047	146138	2423	12249

In both condensed detachment problems Sem_res was able to generate a proof using far fewer given and generated clauses than simple binary resolution. The smaller numbers indicate a more efficient search. This is due to the guidance provided by the semantic information from Finder.

These results were possible even with only the limited guidance Sem_res was then using. However, the result for the syntactic problem highlights one of the weaknesses of DSR. In this case it took many more clauses to generate a proof than unaided binary resolution. This probably indicates that the model created by Finder not only gave poor guidance but that it gave 'misleading' guidance. That is, Sem_res began examining the wrong part of the search space based on the semantic information produced by Finder. The possibility of such 'misleading' information is an inherant weakness of an arbitrary model, particularly when only a single model is being used.

Final Results

The improved version of Sem_res exactly implements Slaney's algorithm. The filtering out of clauses that had resolved on disallowed literals was put in place. This was made easier by the ordering maintained by Otter. Otter keeps the literals from the given clause at the beginning of any resolvent clauses. In combination with the number of literals in the given clause chosen from the passive set this information allowed Sem_res to discard clauses that had resolved on a clause picked up from a satellite.

These improvements lead to a large increase in performance for Sem_res. In some cases DSR was able to outperform hyper-resolution in the number of given clauses needed to produce a proof. Given the efficiency of hyper-resolution this is a very encouraging result. However, as table 2 indicates, these results are not independent of the number of literals in the clauses output by Otter. As the maximum number of literals allowed in the output increases, the number of given clauses needed for Sem_res to find a proof also increases. In contrast, the performance of hyper-resolution is independent of the number of literals in the generated clauses.

Table 2: The comparative results of Dynamic Semantic Resolution

		Resolution Type		
Problem	**Statistic**	**Hyper**	**Binary**	**Dynamic Semantic**
Condensed Detachment 1 (Max Literals = 2)	Given Clauses	63	906	29
	No. of Inferences	63	906	60
	Generated Clauses	2593	70 763	127
	Kept Clauses	439	50 266	69
Condensed Detachment 1 (Max Literals = 3)	Given Clauses	63	1242	66
	No. of Inferences	63	1242	144
	Generated Clauses	2593	234 215	720
	Kept Clauses	439	176 277	171
Condensed Detachment 2 (Max Literals = 2)	Given Clauses	64	667	37
	No. of Inferences	64	667	77
	Generated Clauses	2669	50 209	171
	Kept Clauses	458	35 738	78
Condensed Detachment 2 (Max Literals = 3)	Given Clauses	64	915	64
	No. of Inferences	64	915	143
	Generated Clauses	2669	161 960	719
	Kept Clauses	448	121 768	151
Puzzle Problem 1 (Max Literals = 8)	Given Clauses	47	806	154
	No. of Inferences	47	806	2671
	Generated Clauses	39	13 479	6716
	Kept Clauses	34	2623	2736
Puzzle Problem 2 (Max Literals = 8)	Given Clauses	14	29	9
	No. of Inferences	14	29	56
	Generated Clauses	11	45	34
	Kept Clauses	23	27	33
Syntactic Problem (Max Literals = 4)	Given Clauses	128	925	1575
	No. of Inferences	128	925	5594
	Generated Clauses	527	109 858	11089
	Kept Clauses	116	1948	4281
Syntactic Problem (Max Literals = 5)	Given Clauses	128	1046	527
	No. of Inferences	128	1046	3595
	Generated Clauses	527	146 253	9503
	Kept Clauses	116	3233	3424

The other point to note about table 2 is that it doesn't show cases where Sem_res was unable to find a proof given output clauses restricted to a certain number of literals. This ocurred with Puzzle Problem 1 when a maximum of 7 literals was allowed. This may indicate that the arbitrary nature of the model and the discarding of clauses which are true in it leads to a problem with completeness.

CONCLUSION

The results on the test problems that Sem_res has been given indicate the potential of dynamic semantic resolution as a rule of inference. In the best case it was able to outperform hyper-resolution in the number of given clauses required to generate a proof. This was achieved without any tuning of Otter's parameters. One potential extension of the work presented here would be to integrate DSR directly into Otter. This would allow a more direct comparison between DSR and the other rules of inference. This in turn would allow some quantification of the expense of generating models in Finder.

The results also highlighted the weaknesses of DSR, namely poor guidance and incompleteness. Further testing is needed to gauge the full extent of the difficulties. However, The use of more than one model and some kind of semantic weighting rather than the discarding of clauses could be used to overcome, or minmise, these problems.

ACKNOWLEDGEMENTS

I am grateful to John Slaney for his supervision during the development of the Sem_res program. I am also grateful to the Research School of Information Sciences and Engineering at the ANU in Canberra whose Summer Research Scholarship funded the initial component of the project.

REFERENCES

[CL73] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[Rob65a] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965. Reprinted in [Rob83a].

[Rob65b] J. A. Robinson. Automatic Deduction with Hyper-Resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965. Reprinted in [Rob83b].

[Rob83a] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning: Classical Papers on Computational Logic*, volume 1: 1957-1966, pages 397–415. Springer, 1983.

[Rob83b] J. A. Robinson. Automatic deduction with hyper-resolution. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning: Classical Papers on Computational Logic*, volume 1: 1957-1966, pages 416–423. Springer, 1983.

[Sla67] J. R. Slagle. Automatic Theorem Proving With Renamable and Semantic Resolution. *Journal of the Association for Computing Machinery*, 14(4):687–697, 1967. Reprinted in [Sla83].

[Sla83] J. R. Slagle. Automatic Theorem Proving With Renamable and Semantic Resolution. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning: Classical Papers on Computational Logic*, volume 2: 1967-1970, pages 484–489. Springer, 1983.

[Wos88] L. Wos. *Automated Reasoning: 33 Basic Research Problems*. Prentice Hall, 1988.

[WRC65] L. T. Wos, G. A. Robinson, and D. F. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *Journal of the Association for Computing Machinery*, 12(4):536–541, 1965. Reprinted in [WRC83].

[WRC83] L. T. Wos, G. A. Robinson, and D. F. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning: Classical Papers on Computational Logic*, volume 1: 1957-1966, pages 484–489. Springer, 1983.