Experiments with a Linear Backward Chaining Planner

— Working Paper, April 1999, Edited May 2003 —

Christoph Wernhard
wernhard@uni-koblenz.de

Abstract. The performance of an implementation of the *linear backward chaining* planning algorithm is compared to other planning systems by means of the problem set of the *first ai planning systems competition (AIPS-98)*.

1 Introduction

We describe some experiments with an implementation of the *linear backward chaining (LBC)* planning algorithm [Fro96c,Fro96a], which is based on the *linear connection method* [Bib86], on the problem set of the *first ai planning systems competition AIPS-98*.[1] The purpose of these experiments was to get a feeling for the performance of LBC and estimate the potential of this approach by comparing it to the state-of-the-art planning systems which were participating at the competition: *BLACKBOX*, *HSP*, *IPP*, and *STAN*.[2]

Also some oddities of the the *Planning Domain Definition Language (PDDL)*,[3] that has been used at the competition to specify planning problems, became apparent and are noted here.

2 Implementation

2.1 Language and Environment

Our implementation of LBC, which we call *LBCP*, follows the outlines in [Fro96c,Fro96a]. The core system is written in Eclipse Prolog and works by generating Prolog code that performs the actual plan search, very similar to the Prolog Technology Theorem Prover [Sti88].[4]

[1] See `http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html` and `http://www.cs.cmu.edu/~aips98`.

[2] Also *SGP* took part in the competition, but not on problems that are expressible in the languege to which our variant of LBC is confined, i.e. the STRIPS fragment of PDDL.

[3] See `http://cs-www.cs.yale.edu/homes/dvm/`.

[4] In many aspects our implementation is naive, e.g. multisets like `pool` [Fro96b] are represented simply as unordered lists.

In the “linear” approach to planning, which is adopted by LBC, precondition and effect of a rule are *bags* of fluents, i.e. they are constructed by a non-idempotent, “linear”, conjunction operator. Planning problems expressed in the STRIPS fragment of PDDL can in principle be translated to this format. Unfortunately, as mentioned above, PDDL has some semantic oddities, which prevent that this translation can be done fully automatically. This will be discussed in section 3 below.

In our system, the translation is performed — as far as possible – by a Common Lisp program that converts the STRIPS fragment of PDDL into Prolog terms suitable as input to the core planner.

Optionally output plans are generated as Prolog terms during search. Logical variables can serve as labels, such that a plan is represented by a directed acyclic graph, expressing the partial ordering of actions.[5] These plan terms can be converted into input terms of the *DaVinci* graph displaying tool.[6]

2.2 Details of the Implementation

Our implementation uses variants of some refinements which are well known for similar theorem provers [GLMS94]. The *LIP* refinement [Fro96b] which has been designed for LBC, however has not yet been included.

- *Iterative Deepening Bounds* — Both, a depth bound and an inference bound are implemented. The inference bound counts the number of actions involved in the plan, thereby ensuring that a plan with minimal length is returned. Since the depth bound however shows much better performance on the competition problems, the inference bound has not been used in the reported experiments. Variants of both bounds that decrement by the length of the bodies of the entered contrapositives are available too.
- *Subgoal Sorting* — Several modes of subgoal sorting are available.
  - *Heuristic Sorting* — This is performed on each contrapositive separately. It has no firm theoretical basis and is given here just as documentation. With a literal L a tuple of numbers is associated:
    ```
    < IF Some consequent is identical to L      THEN 1 ELSE 0,
      IF L is instance of a variant of the head THEN 1 ELSE 0,
      Number of variables not shared with head,
      IF Predicate of L is a fluent predicate   THEN 1 ELSE 0,
      Number of variables in L >
    ```
    Literals are then sorted according to these tuples, which are compared lexically from left to right.
  - *Random Ordering* — This is performed on each contrapositive separately. The intended purpose of this option is to estimate the effect of heuristic orderings by comparing them to a set of of randomly generated

[5] Construction of plan terms slows down the search: without labels about 6 percent, with labels about 35 percent.

[6] See `http://www.informatik.uni-bremen.de/~davinci`.

orderings. Maybe for some kinds of problems it would make sense to run proof tasks with different random orderings in parallel.

Two further subgoal ordering schemes that have not been used in the reported experiments are available: just preserving the input ordering and a dynamic reordering during the search phase in which among the subgoals of the same clause, those without counterpart in the pool, are preferred.

- *Contrapositive Sorting* — As for subgoal sorting, a heuristic mode and a random mode are available.
- *Disequality Constraints* — In the current implementation those constraints are automatically generated only for "tautological" instantiations of the rules, i.e. instantiations in which the consequent is a sub-multiset of the antecedent. The disequality constraints can either be processed as disequality subgoals merged into contrapositive bodies (like in older versions of SETHEO [LSBB92]), or as actual constraints associated with variables by using the constraint mechanism provided by Eclipse (i.e. `~=/2`).[7]
- *Antilemmas* — For each open subgoal solutions can be stored. If a new solution is subsumed by a stored solution then this solution fails, else a copy of the new solution is put into the store and the solution succeeds. Before the subgoal is left, its stored solutions are deleted. A solution $\langle V, P\rangle$ consists of the values V of variables in the subgoal, and the result pool P. $\langle V_1, P_1\rangle$ is subsumed by $\langle V_2, P_2\rangle$ if there is a substitution σ such that $V_1 = (V_2\sigma)$ and $P_1 \stackrel{\bullet}{\subseteq} (P_2\sigma)$.
- *Removal of Cyclic Contrapositives* — Cyclic contrapositives [Fro96b] can optionally be removed. Completeness is lost when this option is set.

3 Problem Representation

The competition problems used for the experiments were given in the STRIPS fragment of PDDL. A program translates them into input for the LBC planner. Unfortunately this translation cannot be done fully automatically.

3.1 Translation of Rules

PDDL actions can be converted straightforwardly into LBC rules, provided two constraints, which seem not included in the PDDL specification for the STRIPS fragment, are met:

[7] Although the first variant is weaker, since it considers the constraints only within the scope of a single rule body, in practice it generally seems a little faster than the Eclipse constraint mechanism.

In general the Eclipse constraint mechanism can become rather awkward with term copying and comparing terms for subsumption, since special constraint handlers for these cases have to be written. For some library predicates (e.g. `prune_instances/2`) it is not specified, whether they ensure that these handlers are considered.

1. Fluents occurring negated in the consequent of a rule must occur positively in the antecedent.[8]
2. Antecedents contain no negated fluents.

3.2 Fluents and Classical Literals

PDDL does not distinguish between fluents and classical literals. Both of them come mixed together as init-list in a problem definition. Whether a literal is considered as fluent or classical depends on the rules which are applied to it.[9]

To translate PDDL we have to make some inferencing and guessing. We assume that the atoms of literals in PDDL consequents are fluents (this assumption seems safe). We proceed for simplicity by assuming that for fluents and classical facts different predicates are used. This distinction works on the competition problems, but is not desirable in general, as for example a `clear(table)` fact in blocks world problems shows.

3.3 Types

In our translation unary non-fluent literals are assumed to be typing statements.[10] They are processed at compilation time by wrapping their argument expressions with type functors (e.g. `location(city_1_1)` as object term, `at(obj(X), location(Y))` as literal in a rule). Nesting of type functors allows to express simple type hierarchies (e.g. `location(_)`, `location(airport(_))`), which are inferred using the type extents in the input facts of a given PDDL problem.

4 Presentation of Results

We present the results of those 100 problems of the initial problem set which could be solved at least by one of the competition planners along with the results for several configurations of LBCP in tables for the different domains.

The tables contain for each planner and problem the CPU-time in seconds and the number of actions in the returned plan. "–" indicates that the planner failed to solve the problem.

For the competition planners the CPU-time reported in the competition results is used. The platform and whether startup and preprocessing times are included is not known. For LBCP the CPU-time on a Pentium 166 MHz Linux system as determined by Eclipse Prolog is used. The timeout for each problem was 300 seconds. Preprocessing (i.e. generating and compiling the code which

[8] The negated fluents in the consequent make up the "delete list" in PDDL.

[9] Maybe this can complicate planning algorithms, since it makes the order of rule application asymmetric, i.e. application of a rule treating a literal classically can be followed by application of a rule treating it as fluent, but not vice versa.

[10] PDDL provides means to express typing explicitly in other syntax variants than the STRIPS fragment. But the ADL variant used in the competition has a lot of other syntactic constructs besides typing.

performs the search) is included, but not the startup time of the Prolog system nor the Lisp part of the problem translation.

LBCP does not yet include efficient strategies to return plans with optimal cost (i.e. weighted plan size) or time. Since LBCP enumerates plans, with iterative deepening over depth, plans with differing size are usually returned in short succession. The order in which LBCP enumerates plans at a given depth bound depends largely on the ordering of contrapositives. In our experiments we simply stopped LBCP after it returned the first plan and give the size of that plan. The antilemma mechanism is currently used without considering plan size and can block subgoal solutions with a shorter plan than an earlier solution. Removal of cyclic contrapositives, which was used in some of the experiments, can require longer plans than necessary for a problem.

For all experiments iterative deepening over the depth bound was used, since the inference/plan-size bound showed much worse performance on the problems.

Plan term generation without labels (see section 2.1) is turned on in all reported experiments with LBCP.

Since the ordering of subgoals and contrapositives has on one hand a strong influence on the success of a planning task, but on the other hand seems rather arbitrarily, a third column for LBCP is included in the tables to indicate roughly the amount of independence from the orderings or from possible “tuning”. Each problem is repeated — with the original option settings, but with random ordering of subgoals and contrapositives — with 10 different random seeds. How many of these 10 tasks succeeded within a timeout of 5 seconds for each, is given in the third column.

LBCP was used with four different option settings:

– **LBCP-1** LBCP with standard settings: clause-local tautology disequations, heuristic subgoal and contrapositive ordering, iterative deepening over depth.
– **LBCP-2** Like LBCP-1 but with antilemmas.
– **LBCP-3** Like LBCP-1 but with antilemmas and with cyclic contrapositives removed.
– **LBCP-4** Like LBCP-1 except that the iterative deepening is over depth decremented by length of entered contrapositive bodies.

5 Problem Sets

For each problem set the description text of the AIPS-98 competition is quoted, followed by some remarks, if appropriate.

5.1 Gripper

There is a robot with two grippers. It can carry a ball in each. The goal is to take N balls from one room to another; N rises with problem number. Some planners treat the two grippers asymmetrically, giving rise to an unnecessary combinatorial explosion. This domain was created by Jana Koehler.

Like STAN, LBCP-1 and LBCP-2 return short plans in which at each step both grippers of the robot are used simultaneously to transport two balls. Like IPP, LBCP-3 and LBCP-4 return plans in which only a single gripper is used.

5.2 Logistics

There are several cities, each containing several locations, some of which are airports. There are also trucks, which can drive within a single city, and airplanes, which can fly between airports. The goal is to get some packages from various locations to various new locations. This domain was created by Bart Selman and Henry Kautz, based on an earlier domain by Manuela Veloso.

5.3 Movie

In this domain, the goal is always the same (to have lots of snacks in order to watch a movie), but the number of constants increases with problem number. Some planners have combinatorial problems in such cases. This domain was created by Corin Anderson.

Like all the planners in the competition, LBCP solves all of the problems in this set with proofs of length 7 in very short time. We have therefore only included a single representative in the table. This problem set seems — in the given formulation — rather obscure. The rule `rewind_movie` deletes `counter_at_zero` which is not in the antecedent, thereby violating constraint 1 of section 3.1. The rule `rewind_movie_2` is redundant since it has an antecedent `counter_at_two_hours` that doesn't appear anywhere else in the rules or problems. Under the assumption that any type has at least one member, the arguments of the `get_...`-rules don't play any role at all, and the enumeration of members in the input facts is redundant.

5.4 Grid

There is a square grid of locations. A robot can move one grid square at a time horizontally and vertically. If a square is locked, the robot can move to it only by unlocking it, which requires having a key of the same shape as the lock. The goal is to get keys from various locations to various new locations. This domain was created by Jana Koehler, based on an earlier domain by Drew McDermott.

5.5 Mystery

There is a planar graph of nodes. At each node are vehicles, cargo items, and some amount of fuel. Objects can be loaded onto vehicles (up to their capacity), and the vehicles can move between nodes; but a vehicle can leave a node only if there is a nonzero amount of fuel there, and the amount decreases by one unit. The goal is to get cargo items from various nodes to various new nodes. To disguise the domain, the nodes were called emotions, the cargo items were *pains, the vehicles were pleasures, and fuel and capacity numbers were encoded as geographical entities. This domain was created by Drew McDermott.*

The PDDL input facts for these problems contain a type predicate `province/1` which seems to stem from the ADL problem formulation but is omitted in the rules of STRIPS version. We have dropped these facts.

5.6 Mprime

This is the mystery domain with one extra action, the ability to squirt a unit of fuel from any node to a neighboring node, provided the originating node has at least two units. Created by Drew McDermott.

Like in *mystery*, we have dropped the `province/1` facts here.

Problem	Blackbox		HSP		IPP		STAN		LBCP-1			LBCP-2			LBCP-3			LBCP-4		
strips-gripper-x-1	0.1	11	2.0	13	0.0	15	0.0	11	0.1	11	10	0.1	11	10	0.1	15	4	0.1	15	10
strips-gripper-x-2	7.8	17	2.2	21	0.4	23	1.1	17	0.1	17	10	0.1	17	10	0.1	23	6	0.2	23	10
strips-gripper-x-3	–		2.5	31	3.3	31	54.7	23	0.1	23	10	0.2	23	10	0.1	31	6	0.1	31	10
strips-gripper-x-4	–		3.1	37	26.7	39	3038.4	29	0.1	29	10	0.2	29	10	0.1	39	5	0.2	39	10
strips-gripper-x-5	–		3.3	47	226.5	47	–		0.1	35	9	0.2	35	6	0.2	47	4	0.2	47	10
strips-gripper-x-6	–		3.8	53	–		–		0.1	41	7	0.2	41	5	0.2	55	7	0.2	55	10
strips-gripper-x-7	–		4.8	63	–		–		0.2	47	5	0.3	47	4	0.2	63	7	0.2	63	10
strips-gripper-x-8	–		5.6	71	–		–		0.2	53	9	0.3	53	8	0.2	71	5	0.3	71	10
strips-gripper-x-9	–		6.7	79	–		–		0.2	59	8	0.4	59	4	0.2	79	6	0.3	79	10
strips-gripper-x-10	–		7.6	85	–		–		0.3	65	8	0.5	65	5	0.2	87	6	0.3	87	10
strips-gripper-x-11	–		9.1	93	–		–		0.3	71	4	0.6	71	2	0.2	95	4	0.4	95	10
strips-gripper-x-12	–		10.6	101	–		–		0.4	77	2	0.7	77	1	0.3	103	8	0.4	103	9
strips-gripper-x-13	–		12.5	109	–		–		0.5	83	4	0.8	83	4	0.3	111	6	0.5	111	9
strips-gripper-x-14	–		15.0	119	–		–		0.5	89	6	1.0	89	6	0.3	119	8	0.5	119	9
strips-gripper-x-15	–		16.9	125	–		–		0.6	95	4	1.2	95	4	0.4	127	7	0.6	127	8
strips-gripper-x-16	–		20.1	135	–		–		0.7	101	4	1.5	101	4	0.4	135	5	0.6	135	8
strips-gripper-x-17	–		23.6	143	–		–		0.8	107	5	1.8	107	5	0.4	143	5	0.7	143	6
strips-gripper-x-18	–		27.0	151	–		–		0.9	113	5	2.1	113	5	0.4	151	5	0.8	151	5
strips-gripper-x-19	–		29.9	157	–		–		1.1	119	2	2.5	119	2	0.5	159	6	0.8	159	3
strips-gripper-x-20	–		33.2	165	–		–		1.2	125	5	3.0	125	4	0.5	167	5	0.9	167	5
strips-log-x-1	2.1	27	79.7	43	0.9	26	0.8	27	–		0	–		0	165.6	47	0	–		0
strips-log-x-2	6.4	32	97.1	44	–		4.3	32	1.4	39	0	1.4	39	0	2.3	36	0	173.0	39	0
strips-log-x-5	–		14.4	26	2.4	24	364.9	29	1.2	29	2	0.9	29	3	0.8	26	1	4.2	29	1
strips-log-x-7	–		788.9	112	–		–		–		0	–		0	–		0	–		0
strips-log-x-11	6.5	32	86.2	30	6.9	33	12.8	34	–		0	–		0	–		0	–		0
strips-log-y-1	0.5	13	4.3	13	0.2	13	0.1	13	0.2	13	5	0.2	13	8	0.2	13	2	2.6	13	1
strips-log-y-2	0.6	20	5.3	27	0.3	20	0.2	20	–		0	–		0	1.2	26	2	–		0
strips-log-y-3	0.9	27	10.5	32	0.6	27	0.4	27	–		0	33.3	40	0	4.0	39	2	77.0	33	0
strips-log-y-4	–		170.4	59	–		–		–		0	–		0	69.7	73	0	–		0
strips-log-y-5	–		–		66.2	31	–		–		0	28.5	49	0	3.2	35	0	–		0
strips-movie-x-30	0.1	7	3.1	7	0.0	7	0.1	7	0.1	7	10	0.1	7	10	0.1	7	10	0.1	7	10
strips-grid-y-1	12.2	14	9.3	24	4.6	20	2.5	14	–		0	–		0	–		0	–		0
strips-grid-y-2	–		–		13.9	31	–		–		0	–		0	–		0	–		0
strips-grid-y-4	–		–		84.3	47	–		–		0	–		0	–		0	–		0

Table 1: Results for Gripper, Logistics, Movie and Grid

Problem	Blackbox		HSP		IPP		STAN		LBCP-1			LBCP-2			LBCP-3			LBCP-4		
strips-mysty-x-1	0.1	5	2.2	5	0.1	5	0.0	5	0.3	6	7	0.4	6	4	0.3	6	8	0.3	5	7
strips-mysty-x-2	6.1	9	67.3	22	12.5	9	0.4	9	2.0	11	4	3.6	11	3	1.2	11	6	2.4	9	5
strips-mysty-x-3	0.6	4	3.9	4	1.2	4	0.2	4	0.3	4	7	0.3	4	6	0.3	4	8	0.3	4	6
strips-mysty-x-9	1.4	8	7.3	8	1.9	8	0.2	8	–		2	–		2	–		3	3.3	12	4
strips-mysty-x-10	–		–		–		9.2	8	–		0	–		0	–		0	140.2	9	0
strips-mysty-x-11	0.7	7	4.5	12	0.4	7	0.1	7	0.4	10	5	0.5	10	5	0.4	10	7	0.4	10	3
strips-mysty-x-13	–		–		–		1.8	16	–		0	–		0	–		0	–		0
strips-mysty-x-17	3.5	4	86.9	10	20.6	4	1.7	4	1.7	6	2	2.4	6	2	1.4	6	3	1.3	6	3
strips-mysty-x-19	7.6	8	–		14.9	9	0.9	8	2.2	7	2	3.1	7	2	1.8	7	2	1.4	7	2
strips-mysty-x-25	0.1	4	2.3	4	0.1	4	0.0	6	0.2	5	10	0.2	5	9	0.2	5	8	0.2	5	9
strips-mysty-x-26	1.7	8	11.3	13	2.1	6	0.7	10	2.0	9	0	2.6	9	0	1.9	9	2	2.6	9	0
strips-mysty-x-27	0.6	7	4.3	5	1.0	7	0.3	7	0.2	5	6	0.3	5	5	0.3	5	9	0.2	5	6
strips-mysty-x-28	0.5	7	2.7	18	0.3	7	0.1	7	0.2	10	8	0.3	10	6	0.3	10	8	0.3	10	9
strips-mysty-x-29	0.5	4	3.4	5	0.9	4	0.1	4	0.4	8	6	0.6	8	6	0.4	8	9	0.3	4	7
strips-mysty-x-30	5.6	12	100.1	18	10.0	13	7.7	13	1.4	11	2	1.7	11	1	1.5	11	1	1.2	11	4
strips-mprime-x-1	0.9	9	3.7	5	3.5	11	2.5	11	0.5	7	3	0.7	7	2	0.7	7	4	0.6	7	3
strips-mprime-x-2	6.5	9	110.1	12	21.9	9	2.8	13	2.6	13	3	4.4	13	3	1.9	13	6	5.4	11	3
strips-mprime-x-3	0.9	4	5.6	4	6.3	5	10.2	6	0.3	4	4	0.3	4	4	0.3	4	8	0.3	4	4
strips-mprime-x-4	0.8	9	3.2	10	3.7	12	–		32.0	20	0	23.2	20	0	20.1	20	0	29.8	20	0
strips-mprime-x-5	–		–		10.4	11	1.3	15	31.4	16	0	52.4	16	0	34.5	16	0	7.2	16	0
strips-mprime-x-7	–		–		5.2	5	1.3	6	3.4	10	1	6.8	10	1	1.7	10	2	1.6	10	2
strips-mprime-x-8	2.8	7	23.8	7	32.0	10	–		11.1	12	1	15.6	12	0	13.2	12	0	–		1
strips-mprime-x-9	1.9	8	9.4	8	7.3	9	2.0	16	126.9	18	1	105.0	18	0	78.0	18	1	27.3	20	2
strips-mprime-x-11	1.8	8	7.3	19	6.2	8	7.0	17	0.8	11	2	1.3	11	1	1.1	11	3	1.0	11	0
strips-mprime-x-12	2.3	10	51.0	10	8.7	9	–		1.3	8	2	1.7	8	2	1.4	8	2	1.2	7	3
strips-mprime-x-16	5.2	11	207.4	13	–		–		1.2	12	4	1.6	12	4	1.2	12	6	1.4	6	4
strips-mprime-x-17	7.1	4	608.5	4	–		–		4.1	6	2	5.8	6	2	4.6	6	1	3.8	6	3
strips-mprime-x-21	–		–		–		1.1	7	2.8	8	2	6.1	8	2	1.3	8	2	1.1	6	2
strips-mprime-x-25	0.1	4	2.5	4	1.4	4	0.3	6	0.2	6	5	0.2	6	5	0.2	6	8	0.2	6	4
strips-mprime-x-26	3.2	8	23.9	8	19.2	7	–		–		1	–		0	–		2	–		1
strips-mprime-x-27	3.2	8	67.4	8	–		–		0.3	5	6	0.3	5	5	0.3	5	7	0.3	5	6
strips-mprime-x-28	2.0	11	79.7	7	7.0	10	3.0	14	0.3	12	6	0.5	12	4	0.4	12	8	0.4	12	7
strips-mprime-x-29	2.3	5	24.1	6	–		13.5	7	0.7	8	6	1.1	8	6	0.7	8	8	0.3	5	7
strips-mprime-y-1	0.9	5	7.3	4	4.1	5	0.9	5	0.5	5	3	0.7	5	3	0.4	5	6	0.4	5	4
strips-mprime-y-2	2.5	7	12.8	8	2.7	8	4.8	9	4.1	12	2	6.9	12	2	2.1	12	2	0.8	12	2
strips-mprime-y-4	1.7	5	8.4	4	11.4	4	0.4	5	0.6	5	2	1.2	5	2	0.6	5	7	0.5	5	4
strips-mprime-y-5	0.5	6	4.6	6	2.9	6	–		0.3	6	4	0.4	6	3	0.3	6	5	0.3	6	3

Table 2: Results for Mystery and Mprime

References

[Bib86] Wolfgang Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[Fro96a] Bertram Fronhöfer. Cutting connections in linear connection proofs. In *Int. Computer Symposium '96*, pages 109–117, December 1996.

[Fro96b] Bertram Fronhöfer. Cutting connections in linear connection proofs. Technical Report AR-96-01, Research Group Automated Reasoning, TU München, 1996.

[Fro96c] Bertram Fronhöfer. Situational calculus, linear connection proofs and STRIPS-like planning: an experimental comparison. In P. Migliolo, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of the 5th International Workshop on Theorem Proving with analytic Tableaux and Related Methods (TABLEAUX)*, volume 1071 of *LNAI*, pages 193–209, Berlin, May15–17 1996. Springer.

[GLMS94] C. Goller, R. Letz, K. Mayr, and J. M. P. Schumann. SETHEO V3.2: Recent developments. *Lecture Notes in Computer Science*, 814:778–??, 1994.

[LSBB92] Reinhold Letz, Johann Schumann, S. Bayerl, and Wolfgang Bibel. SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, April 1992.

[Sti88] Mark E. Stickel. A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.