This paper appears in *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (IJCAI-97), Nagoya, Aichi, Japan, August 23–29, 1997, pp. 1169-1176.

Automatic SAT-Compilation of Planning Problems

Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld*
Department of Computer Science and Engineering
University of Washington, Box 352350 Seattle WA 98195–2350 USA
{mernst, todd, weld}@cs.washington.edu

Abstract

Recent work by Kautz *et al.* provides tantalizing evidence that large, classical planning problems may be efficiently solved by translating them into propositional satisfiability problems, using stochastic search techniques, and translating the resulting truth assignments back into plans for the original problems. We explore the space of such transformations, providing a simple framework that generates eight major encodings (generated by selecting one of four action representations and one of two frame axioms) and a number of subsidiary ones. We describe a fully-implemented compiler that can generate each of these encodings, and we test the compiler on a suite of STRIPS planning problems in order to determine which encodings have the best properties.

1 Introduction

Despite the early formulation of planning as theorem proving [Green, 1969], most researchers have long assumed that special-purpose planning algorithms are necessary for practical performance. However, recent improvements in the performance of propositional satisfiability methods [Cook and Mitchell, 1997] cast doubt on this conclusion. Initial results for compiling bounded-length planning problems to SAT were unremarkable [Kautz and Selman, 1992], but recent experiments [Kautz and Selman, 1996] suggest that compilation to SAT might yield the world's fastest STRIPS-style planner.

However, several open questions must be answered before concluding that SAT-based planning dominates specialized algorithms. The experiments of [Kautz and Selman, 1996] used hand-crafted SAT encodings, and while [Kautz *et al.*, 1996] describe methods for compilation, no one has reported experiments on automatically compiled problems and no one knows which encodings are best. The encodings used by [Kautz and Selman, 1996] included domain information that is inexpressible in the STRIPS action language (*e.g.*, the fluent `On` is irreflexive and noncommutative); to what extent is this information responsible for the speedup they observed? This paper addresses these issues:

- We present an analytic framework that accounts for all previously reported non-causal encodings,[1] including several novel possibilities. We parameterize the space of encodings along two major dimensions, action and frame representation. For twelve points in this two-dimensional space, we list the axioms necessary for a minimal encoding, and we calculate the asymptotic encoding sizes.
- We describe an automatic compiler that generates all of these encodings. While it is difficult for a compiler to produce encodings that are as lean as the hand-coded versions of [Kautz and Selman, 1996], we describe type-analysis and factoring techniques that get us close. Experiments demonstrate these methods can reduce the number of variables by half and formula size by 80%.
- We run the compiler on a suite of STRIPS-style planning problems, determining that the regular and simply-split explanatory encodings are smallest and can be solved fastest.

2 The Space of Encodings

This section presents a framework that describes all of the AT&T encodings (except for the causal encodings) as well as some new alternatives. Previous work has described individual encodings in a variety of ways (*e.g.*, "direct," "state-based," *etc.*), but we avoid these terms. Instead we present a parameterized space with two dimensions:

- The choice of a regular, simply split, overloaded split, or bitwise *action representation* specifies the correspondence between propositional variables and ground (fully-instantiated) plan actions. These choices represent different points in the tradeoff between the number of variables and the number of clauses in the formula.
- The choice of classical or explanatory *frame axioms* varies the way that stationary fluents are constrained.

Our encodings use a standard fluent model in which time takes nonnegative integer values. State-fluents occur at even-numbered times and actions at odd times. All of the encodings use the following set of *universal axioms*:

INIT The initial state is completely specified at time zero, including all properties presumed false by the closed-world assumption.

GOAL In order to test for a plan of length n, all desired goal properties are asserted to be true at time $2n$.

*This research was funded in part by Office of Naval Research Grant N00014-94-1-0060, by National Science Foundation Grant IRI-9303461, by ARPA / Rome Labs grant F30602-95-1-0024, and by a gift from Rockwell International Palo Alto Research.

[1] The omitted "state-based" encodings can be obtained by resolving away the actions in our encodings [Kautz *et al.*, 1996].

A⇒P,E Actions imply their preconditions and effects. For each odd time t between 1 and $2n-1$ and for each consistent ground action, an axiom asserts that execution of the action at time t implies that its effects hold at $t+1$ and its preconditions hold at $t-1$. For example, suppose that the initial conditions specify four blocks A, B, C, and D. The STRIPS operator of Figure 1 is inconsistent when instantiated with $o = \texttt{A}$ and $s = \texttt{A}$, but with $o = \texttt{A}$, $s = \texttt{B}$, and $d = \texttt{C}$ it yields the axioms shown, and analogous axioms for preconditions.

2.1 Action Representation

The first major encoding choice is whether to represent actions as regular, simply split, overloaded split, or bitwise. In the **regular** representation, each ground action is represented by a different logical variable, for a total of $\mathcal{A} = n|Ops||Dom|^{A_o}$ such variables (Figure 2 defines these symbols). Since systematic solvers take time exponential in the number of variables, and large numbers of variables also slow stochastic solvers, we would like to reduce this number.

In order to do this, [Kautz and Selman, 1996] introduced **simple operator splitting**, which replaces each n-ary action fluent with n unary fluents throughout the encoding. For example, $\texttt{Move(A,B,C,}t\texttt{)}$ is replaced with the conjunction of $\texttt{MoveArg1(A,}t\texttt{)}$, $\texttt{MoveArg2(B,}t\texttt{)}$, $\texttt{MoveArg3(C,}t\texttt{)}$. Doing this for all fully-instantiated actions reduces the number of variables needed to represent all actions to $n|Ops||Dom|A_o$.

In simple splitting, only instances of the same operator share propositional variables. An alternative is **overloaded splitting**, whereby all operators share the same split fluents. Overloaded splitting replaces $\texttt{Move(A,B,C,}t\texttt{)}$ by the conjunction of $\texttt{Act(Move,}t\texttt{)}$, $\texttt{Arg1(A,}t\texttt{)}$, $\texttt{Arg2(B,}t\texttt{)}$, $\texttt{Arg3(C,}t\texttt{)}$, while a different action $\texttt{Paint(A,Red,}t\texttt{)}$ is replaced with $\texttt{Act(Paint,}t\texttt{)}$, $\texttt{Arg1(A,}t\texttt{)}$, $\texttt{Arg2(Red,}t\texttt{)}$. This technique further reduces the number of variables needed to represent all actions to $n(|Ops| + |Dom|A_o)$.

The **bitwise** representation shrinks the number of variables even more, by representing the actions with only $\lceil \log_2 \mathcal{A} \rceil$ propositional symbols (per odd time step), each representing a bit. The ground actions are numbered from 0 to $\mathcal{A} - 1$. The number encoded by the bit symbols determines the ground action which executes at each odd time step. For instance, if there were four ground actions, then $(\neg\texttt{bit1(}t\texttt{)} \wedge \neg\texttt{bit2(}t\texttt{)})$ would replace the first action, $(\neg\texttt{bit1(}t\texttt{)} \wedge \texttt{bit2(}t\texttt{)})$ would replace the second, and so forth.

2.2 Frame Axioms

FRAME Classical or explanatory frame axioms constrain unaffected fluents when an action occurs.

Classical frame axioms [McCarthy and Hayes, 1969] state what fluents are left unchanged by a given action. For example, one classical frame axiom for the Move operator in Figure 1 would say "Moving block A from B to C leaves D's clearness unchanged," *e.g.*,

$$\texttt{Clear(D,}t{-}1\texttt{)} \wedge \texttt{Move(A,B,C,}t\texttt{)} \Rightarrow \texttt{Clear(D,}t{+}1\texttt{)}$$

Adding classical frame axioms for each action and each odd time t to the universal axioms almost produces a valid encoding of the planning problem. However, if no action occurs at time t, the axioms of the encoding can infer nothing about the truth value of fluents at time $t+1$, which can therefore take on arbitrary values. The solution is to add AT-LEAST-ONE axioms for each time step.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\texttt{Move(}o,s,d\texttt{)}} \\
 & \text{PRECOND}: & \texttt{Block(}o\texttt{)} \wedge \texttt{Clear(}o\texttt{)} \wedge \\
 & & (\texttt{Table(}d\texttt{)} \vee \texttt{Clear(}d\texttt{)}) \wedge \\
 & & \texttt{On(}o,s\texttt{)} \wedge o \neq s \wedge o \neq d \wedge s \neq d \\
 & \text{EFFECT}: & \texttt{Clear(}s\texttt{)} \wedge \neg\texttt{On(}o,s\texttt{)} \wedge \\
 & & \neg\texttt{Clear(}d\texttt{)} \wedge \texttt{On(}o,d\texttt{)}
\end{array}
$$

$$
\begin{array}{l}
\texttt{Move(A,B,C,}t\texttt{)} \Rightarrow \texttt{Clear(B,}t{+}1\texttt{)} \\
\texttt{Move(A,B,C,}t\texttt{)} \Rightarrow \neg\texttt{On(A,B,}t{+}1\texttt{)} \\
\texttt{Move(A,B,C,}t\texttt{)} \Rightarrow \neg\texttt{Clear(C,}t{+}1\texttt{)} \\
\texttt{Move(A,B,C,}t\texttt{)} \Rightarrow \texttt{On(A,C,}t{+}1\texttt{)}
\end{array}
$$

Figure 1: Top: STRIPS definition of a blocks-world operator for moving an object from a source to a destination. Bottom: Axiom schema showing an instance of Move implies its effects.

$\lvert Ops\rvert$	number of operators
$\lvert Pred\rvert$	number of predicate symbols
$\lvert Dom\rvert$	number of constants in the domain
n	number of odd time steps in plan (may be < plan length)
A_p	max arity of predicates
A_o	max arity of operators
A_r	length of action representation (predicate symbols per action): regular = 1; simple split = A_o; overloaded split = $A_o + 1$; bitwise = $\lceil \log_2 \mathcal{A} \rceil$
$\mathcal{A}$	$= \lvert Ops\rvert \lvert Dom\rvert^{A_o}$ number of ground actions
$\mathcal{F}$	$= \lvert Pred\rvert \lvert Dom\rvert^{A_p}$ number of ground fluents
P_o	$= O(\mathcal{F})$ max num fluents mentioned in operator

Figure 2: Symbols used in complexity analyses.

AT-LEAST-ONE A disjunction of every possible fully-instantiated action ensures that some action occurs at each odd time step. (A no-op action is inserted as a preprocessing step.) Note that action representation has a huge effect on the size of these axioms (Figure 3).[2]

The resulting plan consists of a totally-ordered sequence of actions; indeed it corresponds roughly to a "linear" encoding in [Kautz *et al.*, 1996], except that they include exclusion axioms (see below) to ensure that at most one action is active at a time. However, exclusion axioms are unnecessary because the classical FRAME axioms combined with the A⇒P,E axioms ensure that any two actions occurring at time t lead to an identical world-state at time $t+1$. Therefore, if more than one action does occur in a time step, then either one can be selected to form a valid plan.

Explanatory frame axioms [Haas, 1987] enumerate the set of actions that could have occurred in order to account for a state change. For example, an explanatory frame axiom would say which actions could have caused D's clearness status to change from true to false.

$$\texttt{Clear(D,}t{-}1\texttt{)} \wedge \neg\texttt{Clear(D,}t{+}1\texttt{)} \Rightarrow (\texttt{Move(A,B,D,}t\texttt{)} \vee \texttt{Move(A,C,D,}t\texttt{)} \vee \ldots \vee \texttt{Move(C,Table,D,}t\texttt{)})$$

As a supplement to the universal axioms, explanatory frame axioms must be added for each ground fluent and each odd time t to produce a reasonable encoding. With explanatory frames, a change in a fluent's truth value implies that some action occurs, so (contrapositively) if no action occurs at a time step, this will be correctly treated as a no-op. Therefore, no AT-LEAST-ONE axioms are required.

Since explanatory frames do not explicitly force the fluents not affected by an executing action to remain unchanged,

[2] AT-LEAST-ONE axioms are not necessary if the bitwise action representation is used, because all spare bit patterns can be used to refer to actual ground actions.

Axiom	Action Representation	Clauses	Clause size
INIT	All	$\mathcal{F}$	1
GOAL	All	arbitrary formula, typically small	
A⇒P,E	All	$O(nP_o\mathcal{A})$	A_r+1
FRAME	Classical	$O(n\mathcal{F}\mathcal{A})$	A_r+2
	Explanatory	$O(n\mathcal{F}{A_r}^{\mathcal{A}})$	$O(\mathcal{A})$
AT-LEAST-ONE	Simple factored	$O(n)$	$\lvert Ops\rvert\lvert Dom\rvert$
	Overloaded factored	$O(n)$	$\lvert Ops\rvert$
	All other representations	$O(n{A_r}^{\mathcal{A}})$	$\mathcal{A}$
EXCLUSION	Simple factored	$O(n\lvert Ops\rvert(\lvert Ops\rvert+A_o-1)\lvert Dom\rvert^2)$	2
	Overloaded factored	$O(n(\lvert Ops\rvert^2+A_o\lvert Dom\rvert^2))$	2
	All other representations	$O(n(A_r\mathcal{A})^2)$	2
NO-PARTIAL	Simple Factored:	$O(n\lvert Ops\rvert\lvert Dom\rvert A_o)$	$\lvert Dom\rvert+1$
	Overloaded Factored:	$O(n\lvert Dom\rvert(A_o+1))$	$\lvert Dom\rvert+1$

Figure 3: The sizes of each axiom schema as a function of action representation. Note that combinations whose entries are identical may have different sizes because the value of A_r is itself a function of action representation (see Figure 2).

they permit parallelism. Specifically, any actions whose preconditions are satisfied at time t and whose effects do not contradict each other might be executed in parallel. This kind of parallelism is problematic because it can create valid plans which have no linear solution. For example, suppose action α has precondition X and effect Y, while action β has precondition $\neg Y$ and effect $\neg X$. While these actions might be executed in parallel (because their effects are not contradictory) there is no legal total ordering of the two actions, which is problematic for non-instantaneous real-world actions.

EXCLUSION Linearizability of resulting plans is guaranteed by restricting which actions may occur simultaneously.

Two kinds of exclusion enforce different constraints in the resulting plan:

- **Complete** exclusion: For each odd time step, and for all distinct, fully-instantiated action pairs α, β, add clauses of the form $\neg\alpha_t \vee \neg\beta_t$. Complete exclusion ensures that only one action occurs at each time step, guaranteeing a totally-ordered plan.
- **Conflict** exclusion: For each odd time step, and for all distinct, fully-instantiated, conflicting action pairs α, β, add clauses of the form $\neg\alpha_t \vee \neg\beta_t$. In our framework, two actions conflict if one's precondition is inconsistent with the other's effect.[3] Conflict exclusion results in plans whose actions form a partial order. Any total order consistent with the partial order is a valid plan.

Because we wish to consider the minimal encoding corresponding to each choice of action and frame representations, we will assume that conflict exclusion is used whenever possible. Conflict exclusion cannot be exploited when using a split action representation, because splitting causes there not to be a unique variable for each fully-instantiated action. For example, with simple splitting, it would be impossible to have two instantiations of the same operator execute at the same time, because their split fluents would interfere. Overloaded splitting further disallows two instantiations of different operators to execute at the same time.

The bitwise action representation requires no action EXCLUSION axioms. At any time step, only one fully-instantiated action's index can be represented by the bit symbols, so a total ordering is guaranteed.

[3]Contrast our definition of conflict with that of Graphplan [Blum and Furst, 1995] and [Kautz and Selman, 1996]. Unlike Kautz and Selman's parallel encoding, but like their linear one, our encodings have axioms stating that actions imply their effects; their parallel encoding prohibits effect-effect conflicts instead.

3 Optimizing Axioms with Factoring

Eight base encodings are generated by choosing among the regular, simple split, overloaded split, and bitwise action representations and choosing either classical or explanatory frames. Unfortunately, choices that lead to a small number of variables (*i.e.*, the splitting strategies and bitwise) tend to explode the number of clauses or size of each clause. Consider the AT-LEAST-ONE axiom, which is a disjunction of all fully-instantiated actions. Substituting a conjunction of split or bitwise variables for each regular action literal produces a disjunctive normal form formula which blows up exponentially when converted to conjunctive normal form. With simple splitting, this axiom grows[4] from n clauses of size $\mathcal{A}$ to $n{A_o}^{\mathcal{A}}$ clauses of size $\mathcal{A}$ (see Figure 3).

The formula blowup results from blindly substituting a complete conjunction of split variables for each action in the A⇒P,E, FRAME, AT-LEAST-ONE, and EXCLUSION axioms. *Factoring* can dramatically reduce both the number of clauses and their sizes for simple and overloaded splitting. The idea is to use only a subset of the full conjunction for an action whenever possible. Such a partially-instantiated action represents the set of all fully-instantiated actions consistent with it. The bitwise action representation does not admit an easy method of factoring because partial conjunctions of the bit variables are not useful unless a clever action numbering scheme is created.

3.1 Factoring A⇒P,E and FRAME Axioms

The A⇒P,E and FRAME axioms, which relate a single fluent to a single action, can make good use of partial action instantiations. For example, Figure 1 shows the `Move` operator and some of the A⇒P,E axioms for one possible instantiation of the operator. Ordinary simple splitting will transform the first axiom at the bottom of Figure 1 into

$$\texttt{MoveArg1(A,}t\texttt{)} \wedge \texttt{MoveArg2(B,}t\texttt{)} \wedge \texttt{MoveArg3(C,}t\texttt{)} \Rightarrow \texttt{Clear(B,}t\texttt{+1)}$$

A similar axiom is generated for all pairs of constants s and d for which `Move(`s`,B,`d`,`t`)` is a consistent action. Since two of the argument values are irrelevant for this axiom, the simpler axiom `MoveArg2(B,`t`)` $\Rightarrow$ `Clear(B,`t`+1)` can be used instead, eliminating the need to explicitly consider all $\lvert Dom\rvert^2$ values for `MoveArg1` and `MoveArg3`.

[4]The number of logically independent clauses may be substantially smaller than this worst-case bound which results from naive conversion: some clauses may contain duplicated literals, and some clauses may logically imply others. Our implementation eliminates these unnecessary literals and clauses.

	Action representation					
	Regular	Simple		Overloaded		Bitwise
		Unfactored	Factored	Unfactored	Factored	
Vars	$n\mathcal{F}+n\mathcal{A}$	$n\mathcal{F}+n\lvert Ops\rvert A_o\lvert Dom\rvert$	$n\mathcal{F}+n\lvert Ops\rvert A_o\lvert Dom\rvert$	$n\mathcal{F}+n(\lvert Ops\rvert+A_o\lvert Dom\rvert)$	$n\mathcal{F}+n(\lvert Ops\rvert+A_o\lvert Dom\rvert+1)$	$n\mathcal{F}+n\log_2\mathcal{A}$
Classical	AT-LEAST-ONE $O(n\mathcal{F}\mathcal{A})$	AT-LEAST-ONE $O(n\mathcal{F}\mathcal{A}A_o + nA_o{}^{\mathcal{A}}\mathcal{A})$	AT-LEAST-ONE, NO-PARTIAL $O(n\mathcal{F}\mathcal{A}A_o + n\lvert Ops\rvert\lvert Dom\rvert^2A_o)$	AT-LEAST-ONE $O(n\mathcal{F}\mathcal{A}A_o + nA_o{}^{\mathcal{A}}\mathcal{A})$	AT-LEAST-ONE, NO-PARTIAL $O(n\mathcal{F}\mathcal{A}A_o+n\lvert Dom\rvert^2A_o)$	$O(n\mathcal{F}\mathcal{A}\log_2\mathcal{A})$
Explanatory	EXCLUSION $O(n\mathcal{F}\mathcal{A} + n\mathcal{A}^2)$	EXCLUSION $O(n\mathcal{F}A_o{}^{\mathcal{A}} + n(A_o\mathcal{A})^2)$	EXCLUSION, NO-PARTIAL $O(n\mathcal{F}A_o{}^{\mathcal{A}} + n\lvert Ops\rvert^2\lvert Dom\rvert^2A_o)$	EXCLUSION $O(n\mathcal{F}(\mathcal{A}A_o)^2 + n\mathcal{F}A_o{}^{\mathcal{A}}\mathcal{A})$	EXCLUSION, NO-PARTIAL $O(n\mathcal{F}A_o{}^{\mathcal{A}}\mathcal{A} + n\lvert Dom\rvert^2(A_o+\lvert Ops\rvert^2))$	$O(n\mathcal{F}(\log_2\mathcal{A})^{\mathcal{A}})$

Figure 4: Composition and worst case size of the encodings. The bitwise action representation yields the smallest number of variables, but the most clauses; regular actions are the exact opposite. All encodings INIT, GOAL, A⇒P,E, and FRAME axioms. Any additional clauses are noted, and the total size for all clauses is given. The reported numbers are asymptotic numbers of literals (*i.e.*, the product of numbers of clauses and clause sizes).

Factoring A⇒P,E axioms relies on this idea: when relating an action to a fluent, we need only include the parts of the action conjunct pertaining to the arguments that appear in the affected fluent.

The technique extends easily to both classical and explanatory FRAME axioms. Consider the classical frame example given in Section 2.2. Instead of naively splitting `Move(A,B,C,`t`)` into `MoveArg1(A,`t`)` ∧ `MoveArg2(B,`t`)` ∧ `MoveArg3(C,`t`)`, we observe that the source and object of the `Move` are irrelevant and generate

$$\texttt{Clear(D,}t{-}1\texttt{)} \wedge \texttt{MoveArg3(C,}t\texttt{)} \Rightarrow \texttt{Clear(D,}t{+}1\texttt{)}$$

This formula implicitly represents the set of all classical frame axioms relating the `clear`ness of `D` to any `Move` action having `C` as its destination argument.

Note that while the factoring optimization is crucial in practice (see Section 5.5), it is equivalent to ordinary splitting in the worst case. In particular, when the arity of precondition and effect fluents is equal to the arity of the operator, no factoring is possible.

3.2 Factoring EXCLUSION Axioms

Since pairwise exclusion clauses relate actions to other actions (*e.g.*, ¬`Move(A,B,C,`t`)` ∨ ¬`Move(A,B,D,`t`)`) instead of relating actions to fluents, the previous technique cannot be used. Instead, we factor these axioms by noting that, rather than excluding whole actions from occurring simultaneously, we can independently exclude the values of each argument to an action.

For example, factored exclusions of the `Move` operator look like (¬`MoveArg`i`(`a`,`t`)` ∨ ¬`MoveArg`i`(`b`,`t`)`), ranging over all arguments i and distinct constants a and b. This ensures that at most one fully-instantiated `Move` action is active at time t. By doing this for all operators, we ensure that only one instance of each operator is active at time t.

To complete the exclusion, we need to ensure that no two operators have an active instance at time t. This is accomplished by pairwise excluding all possible first arguments of each operator with one another. In other words, we add clauses (¬α`Arg1(`a`,`t`)` ∨ ¬β`Arg1(`b`,`t`)`) for all distinct operators α and β and all (not necessarily distinct) constants a and b. Figure 3 shows how factoring reduces the asymptotic number and size of clauses as compared with unfactored split EXCLUSION axioms.

3.3 Factoring AT-LEAST-ONE Axioms

Without factoring, the AT-LEAST-ONE axiom explodes into an exponential morass during the conversion to CNF. Fortunately, it can be factored very easily, yielding the disjunction of all possible first arguments to all operators, *i.e.*, an axiom of the form: (`Op`$_1$`Arg1(A,`t`)` ∨ `Op`$_1$`Arg1(B,`t`)` ∨ ...∨ `Op`$_2$`Arg1(A,`t`)` ∨ `Op`$_2$`Arg1(B,`t`)` ∨ ...). This axiom now requires only n clauses of size $|Ops||Dom|$, quite a reduction from the unfactored case.

3.4 Preventing Partial Action Execution

The previous three subsections show how to factor each part of the encoding. All three parts rely on the ability to refer to parts of an action instead of always referring to a complete instantiation of an action. However, the underlying assumption is that, whenever any part is instantiated, so is the rest of the action.

For example, we would not want a factored frame clause to have any effect unless a full action implied by that frame was actually being executed at the current time step. Otherwise, the frame could constrain the resulting plan, even though the action referred to by the frame is never fully executed.

NO-PARTIAL We add axioms which state that, whenever any part of an operator is instantiated, so is the rest.

Here are the partial action elimination axioms for the `Move` operator:

$$\begin{aligned}
&(\texttt{MoveArg1(A,}t\texttt{)} \vee \texttt{MoveArg1(B,}t\texttt{)} \vee \ldots) \Leftrightarrow \\
&\quad (\texttt{MoveArg2(A,}t\texttt{)} \vee \texttt{MoveArg2(B,}t\texttt{)} \vee \ldots) \\
&(\texttt{MoveArg1(A,}t\texttt{)} \vee \texttt{MoveArg1(B,}t\texttt{)} \vee \ldots) \Leftrightarrow \\
&\quad (\texttt{MoveArg3(A,}t\texttt{)} \vee \texttt{MoveArg3(B,}t\texttt{)} \vee \ldots)
\end{aligned}$$

These axioms ensure that whenever any split fluent of `Move` is true, then some complete instantiation of `Move` is true. Figure 3 shows the number and size of the resulting clauses.

4 The MEDIC Planner

Following the encodings described above, we have implemented a classical planner which accepts traditional[5] inputs (initial state, goal formula, and STRIPS action schemata) and returns a sequence of actions that will achieve the goal. The MEDIC planner operates by compiling the planning problem into clausal form, solving the SAT problem, and translating the satisfying truth assignment back into actions. Depending on the switch settings, any of the SAT encodings described above can be generated. Thus the MEDIC planner forms a unique testbed for exploring the properties of the different encodings.

The architecture of the planner is shown in Figure 5. Action schemata are parsed using the preprocessor from the

[5]By contrast, the implementation of [Kautz and Selman, 1996] accepts "direct" encodings in a logical constraint language, rather than STRIPS actions.

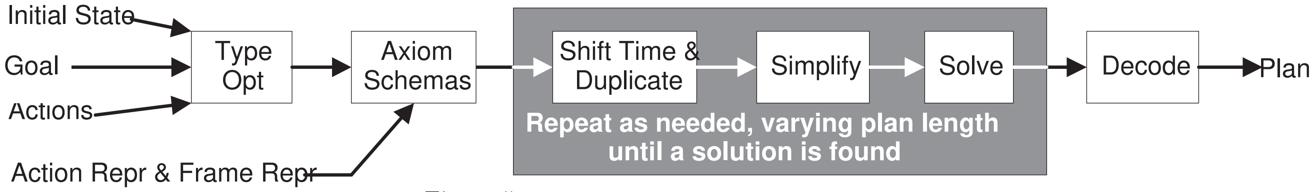

Figure 5: Architecture of the MEDIC planner.

UCPOP planner [Penberthy and Weld, 1992] and type optimization (see below) is performed. Next, guided by the choice of action and frame representations (Figure 4), the compiler creates a master axiom schema representing all action possibilities for one time step. The periodic axiom schema is instantiated multiple times, based on the plan length currently being considered. The output of this duplication module, combined with the initial state and goal specification, is simplified by pure literal elimination, unit clause propagation, and duplicate literal elimination using a fast (linear time) procedure [Van Gelder and Tsuji, 1996]. The resulting clauses are solved using Walksat [Selman *et al.*, 1996] or Tableau [Crawford and Auton, 1993].

4.1 Optimizations

Planning via reduction to propositional satisfiability is impractical without a number of optimizations which determine the truth values of fluents or limit the ground instantiations of actions. Foremost among these are type optimizations. A type is a fluent which no action affects.

Types can constrain operator instantiation by ruling out impossible ground versions. For instance, if `A` and `B` are the only blocks, we can prune any instantiation of the `Move` operator (Figure 1) which does not assign o to either `A` or `B`. When such preconditions are reflected in the operator instantiations, the types themselves need not appear in the final encoding; for instance, the `Block` precondition would be removed from `Move`. This mechanism is a generalization of the obvious one for handling equality and inequality constraints, which are special cases of types.

Because of the usefulness of type information, we have explored methods of inferring types of arguments when operators do not specify them. Suppose that $\texttt{Block}(o)$ did not appear in the Move definition in Figure 1, but that whenever `Clear(o)` appears in an action's effect (for any variable o), that action's precondition contains the fluent $\texttt{Block}(o)$. Then no constant can become `Clear` without being a `Block`. If every constant which is `Clear` in the initial conditions is also a `Block`, we can deduce that every `Clear` constant must be a `Block` and add $\texttt{Block}(o)$ to the `Move` precondition.

Similarly, inequality constraints can be inferred if a fluent appears both positively and negatively in an operator, since the two bindings cannot be identical. Since the `Move` operator of Figure 1 has effects $\neg\texttt{On}(o,s)$ and $\texttt{On}(o,d)$, the $s \neq d$ constraint would be inferred if it were not already present.

An operator's instantiations can be further pruned by eliminating symmetric operator instantiations. For instance, if an operator α takes two arguments which are used identically, then there is no sense considering both of the bindings $\alpha(\texttt{A},\texttt{B})$ and $\alpha(\texttt{B},\texttt{A})$; we arbitrarily select one of the possibilities. This analysis cuts the number of ground instantiations by about an order of magnitude for the refrigerator domain.

The MEDIC planner further reduces bindings and infers invariant fluents by enforcing a form of consistency. An approximation to the set of fluents that can be true (and also to those that can be false) is computed by an iterative dataflow analysis. The first approximation is the initial condition; at each step any fluents in the effects of actions that can fire, given the current approximations, are added to the sets. This process is guaranteed to terminate and is not tantamount to solving the planning problem since time is ignored, thereby permitting impossible situations, like the presence of a fluent and its negation.

The CNF simplification step is also quite important, since it is fast and can reduce the formula size enormously. Though CNF simplification operates without knowledge of the structure of the problem, its effects are similar to some of the optimizations listed above. For instance, it can do much of the type elimination described above. However, performing these steps earlier can reduce encoding time by a factor of four or more due to generation of smaller formulae. Further, these optimizations can often allow the simplifier to reduce a formula more than it otherwise could.

Optimization and Factoring

Factored action representations reduce the benefit of these type optimizations. When performing factored simple splitting, only unary types can be eliminated, since their effect is restricted to (and fully reflected by) just one of the newly-introduced action predicate symbols. Binary types such as $\neq$ cannot be eliminated: consider a binary operator α which takes two non-equal arguments. Given two objects `A` and `B`, only two instantiations $\alpha(\texttt{A},\texttt{B})$ and $\alpha(\texttt{B},\texttt{A})$ are possible, but since the new action fluents $\alpha\texttt{Arg1}$ and $\alpha\texttt{Arg2}$ can each take either `A` or `B` as an argument, it is necessary to leave the axiom $\alpha\texttt{Arg1}(x) \wedge \alpha\texttt{Arg1}(y) \Rightarrow x \neq y$ in the encoding to prevent the illegal argument combinations.

Overloaded action representations do not admit elimination of even unary types, since a single action fluent represents the nth argument to many different operators with different constraints.

4.2 Searching for the Minimal Plan

So far we have assumed that one is trying to find a plan of known length, but in general the plan length is not known in advance. The MEDIC planner is capable of both linear and binary search on plan lengths.[6] Our encodings support the linear search strategy without any modification. To implement binary search for the minimal plan length, we include an explicit no-op (maintain) action when using classical frame axioms. This allows plans longer than the minimal length to succeed.

Because Walksat is stochastic, finding a minimal length plan requires a systematic solver such as Tableau instead of (or in addition to) Walksat. For even moderately-sized problems, however, Tableau can take an unreasonably long time to verify that no solution exists. (Such verification is moot

[6]Because SAT solving time is potentially exponential in encoding size, we conjecture that linear search strategy is better, but we haven't performed serious tests.

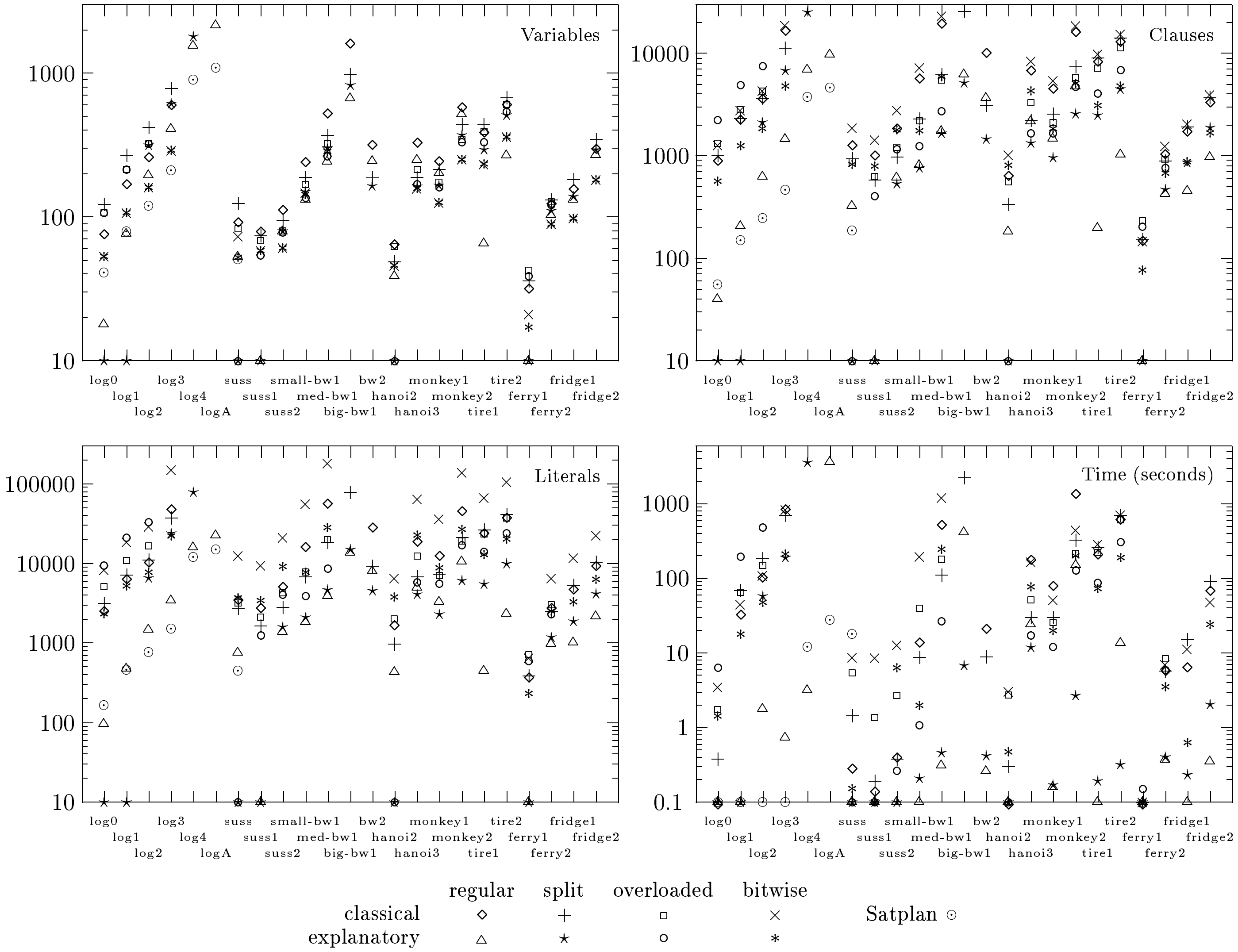

Figure 6: Numbers of variables, clauses, and literals in simplified CNF formulas resulting from each of eight encodings, plus the Satplan hand-encoding (sans domain-specific axioms). Values reported as 10 are actually 0: that is, the CNF simplifier solved the problem. Times less than one tenth second are reported as one tenth.

when trying to find any satisfying plan rather than the shortest one.)

5 Experiments

To test the various encodings, we encoded a suite of planning problems using each of the eight encodings. Factoring was applied when split action representations were used. Figure 6 plots the number of variables, clauses, and literals in the final simplified CNF formulae.

Figure 6 also reports Walksat solution[7] times (averaged over five runs), but note that timing data is hard to interpret. Walksat is not always the fastest solution method. We used the suggested Walksat flag settings from the Satplan planner, but these flags might favor some encodings over others. The timings reported in [Kautz and Selman, 1996] are each minima over many Walksat runs with varying parameter values. It is believed that solution time correlates with CNF size, but automatically determining which solver flags are best for a particular problem is an open problem [Selman *et al.*, 1997], though progress has been made recently [McAllester *et al.*, 1997].

From the asymptotic size bounds of Figure 4 one would expect the bitwise encodings to have the smallest number of variables and the regular encodings to have the largest number of variables. Surprisingly, neither expectation was fulfilled.

5.1 The Smallest Encodings

The two smallest encodings are the regular and simply split explanatory encodings, and these encodings had quick solve times as well. These successes bring to light several interesting points about the relative merits of the encodings.

First, it is clear that explanatory frame clauses are superior to classical frame clauses. Explanatory frames are smaller because they only state what changes, rather than what does not change, when an action occurs. In general, we expect each action to affect relatively few fluents.

[7] We do not report encoding or simplification times, which for medium and large problems are dominated by solution time.

Parallelism is also a big advantage (as shown by the success of the regular explanatory encoding). Since parallel plans have shorter length, the formula contains fewer copies of the periodic axioms. Additionally, conflict exclusion axioms are a subset of complete exclusions, which prohibit all pairs of actions. Conflict exclusion only excludes pairs of actions that would not be otherwise excluded but should be in order to guarantee the existence of a linearization of the partial order plan returned.

It is quite surprising that the regular explanatory encoding has so few variables. [Kautz and Selman, 1996] dismiss this encoding as impractical. While its size can blow up prohibitively in the worst case (see Figure 4), in practice the encoding maintains excellent variable and clause sizes. And it remains competitive even as problems increase in size (*e.g.*, problem sequence log0, log1, ..., logA). We suspect the compiler's type optimizations (which are handicapped by factored splitting) deserve the credit.

5.2 The Largest Encodings

The two worst encodings are the regular and bitwise classical encodings. We have already mentioned the superiority of explanatory to classical frames. Regular classical is outperformed by the two split classical encodings. Worst-case splitting clause sizes can be much bigger than the regular encoding, but in practice factoring seems to keep the sizes competitive. Splitting also may provide the simplifier with more flexibility, allowing it to deduce more, because it can reason about parts of actions instead of only about fully-instantiated actions without hope of generalizing. Finally, these encodings are also aided by the great decrease in the number of variables as compared with the regular encoding.

On the other hand, the bitwise encoding, which has the smallest number of variables before simplification, is the worst encoding of all. Simplification is relatively ineffective on this encoding, as other encodings have fewer variables after the simplification phase. This may be related to the fact that bitwise uses one set of variables to encode all possible actions in the domain, thereby making it next to impossible for the simplifier to reason about the truth values of these variables. Finally, the graph of number of literals points to the obvious blow-up that bitwise incurs in exchange for the small variable size.

5.3 Comparison with Satplan

Although our encodings cannot be expected to be as compact as the hand-made Satplan encodings, our best encodings are surprisingly competitive. The first seven problems of Figure 6 include a ranking for the Satplan direct encoding of the problem, from which domain-specific axioms (see Section 5.6) have been removed for purposes of comparison. Our best encodings actually outperform the Satplan encodings on two of the smaller problems, as the simplification process is able to satisfy our formulas completely. As the problems get larger, the Satplan encodings begin to dominate. However, our best automatic encoding appears to be always within a factor of two of the Satplan size.

5.4 Type Optimizations

Type optimizations can substantially reduce formula size: Figure 7 compares formula sizes with and without these optimizations. These numbers understate the benefits of the optimizations, because they do not include data for problems that were too large to solve without type optimizations

	Regular	Simple	Overloaded	Bitwise
Classical	.31	.39	.40	.32
Explanatory	1.00	.98	.67	.76

Figure 7: Ratio of simplified formula size with type optimizations to simplified formula size without. The numbers reported are averages over seven problems of the ratios for variables, clauses, and literals, which are always within .15 of the average and usually even closer.

	Classical		Explanatory	
	Simple	Overloaded	Simple	Overloaded
Variables	.81	.99	.46	.69
Clauses	.50	.69	.30	.50
Literals	.34	.50	.20	.38

Figure 8: Ratio of simplified formula size with factoring to simplified formula size without.

but could be solved with them. The optimizations are critical for the classical encodings, cutting their size by about two thirds. However, these optimizations are much less effective on explanatory encodings. In fact, the optimizations appear to be superfluous for the regular explanatory encoding: the CNF simplifier obtains all of the type optimization benefits without considering the structure of the problem, using only the resulting formula.

These contrasts may be attributable to the way in which the simplifier interacts with the various encodings. Classical encodings are much more constraining than explanatory encodings, because they explicitly enforce all truth values at time $t+1$ when an action occurs at time t. This rigidity may make it hard for the simplifier to reduce the encoding size, thereby relying more heavily on the type optimizations to make deductions about the encoding. The regular explanatory encoding, which uses conflict exclusion, is the most flexible of all of the encodings. Therefore, it seems that any static optimizations that we make are easily teased out of the encoding by the simplifier.

5.5 Factoring

Figure 8 shows that factoring makes a big difference compared with unfactored splitting. While factoring does not reduce variable size at all in the base encoding, it does lead to small drops in variable size after simplification. Factoring's big effects, however, are in clause and (especially) literal size. This is important, because this reduction is precisely the reason that we introduced the idea of factoring. Although in the worst case, factoring has no effect, it is clear that factoring is critical in practice.

5.6 Domain Specific Axioms

The "direct" encodings of [Kautz and Selman, 1996] provide hand-coded, domain-specific information which is impossible to specify in terms of STRIPS actions but is natural when writing general logical axioms. For example, in their blocks world problems Kautz *et al.* state that the relation `On` is both non-commutative and irreflexive, only one block may be on another at any time, every block is on exactly one other object, blocks can't be both clear and have something on them, and the `Table` is never on anything. To determine how much (if at all) this additional information affected the planning problem, we removed these domain-specific axioms from the AT&T encodings and compared the size and speed of the resulting SAT problems. As Figure 9 shows, eliminating the axioms decreased the number of clauses, but *increased* the number of variables (presumably because unit-clause and

	With domain-specific axioms				Without domain specific axioms			
Problem	**Vars**	**Clauses**	**Time (sec)**	σ	**Vars**	**Clauses**	**Time (sec)**	σ
`bw-large-a`	459	4675	0.97	0.66	534	3060	3.72	2.17
`bw-large-b`	1087	13772	27.18	16.91	1235	7457	71.93	48.60
`bw-large-c`	3016	50457	379.85	505.90	3526	22535	>7000.00	

Figure 9: AT&T's hand-coded domain-specific axioms led to more clauses, fewer variables (after simplification), and substantial speedup. Each problem was run five times on an SGI Indy with Walksat settings: tries 20, noise 30 100, and cutoff set to the number of variables squared. Solve-time standard deviations are reported as σ.

pure-literal simplification was less effective). Without the domain-dependent axioms, the planning problems took substantially longer. These results suggest it would be useful to investigate whether a compiler could deduce some of these axioms automatically. We believe our type optimizations to be a good start at achieving this goal.

6 Conclusions

This paper makes several contributions:

- We develop a simple framework that generates eight major encodings, which account for all of the non-causal AT&T encodings as well as several novel ones. In particular, the introduction of overloaded splitting and the bitwise representation, combined with the regular and simply-split encodings, creates a spectrum of choices highlighting the tradeoff between variable and clause sizes.
- We describe an automatic compiler that takes classical STRIPS planning problems and generates SAT problems using all of the above encodings. Our compiler includes many interesting features, including a type inference and optimization mechanism.
- We use the compiler to perform an empirical analysis of tradeoffs in the space of encodings. We show that explanatory frames and conflict exclusion are dominant, and regular acton representation is surprisingly effective.

Many exciting problems remain. Clearly we need to better investigate the solve-time characteristics of the encodings. Automatically generating domain-specific axioms, such as those in Section 5.6, is a promising direction. We also hope to investigate additional type inference methods. There are also many hybrid encodings which would be interesting to explore. Allowing inter-operator parallelism in the simply-split explanatory encoding could take advantage of both of the best encodings. (As mentioned earlier, simple splitting prevents the possibility of parallel instantiations of the same operator, as their split variables will interfere.) Another hybrid option is the addition of "action" variables, similar to those of overloaded splitting, to the simple splitting encoding. These extra variables can greatly compact many parts of a factored split encoding. A third hybrid would use bitwise representations for the split fluents of simple or overloaded split actions, avoiding the disadvantages of the bitwise action representation while reducing the number of variables. One can also imagine compiling part of a domain theory with one encoding and using a different encoding for other parts. Finally, it would be interesting to automate the AT&T state-based encodings and to integrate their causal encodings into our framework.

Full source code for the Medic planner is available at `ftp://ftp.cs.washington.edu/pub/ai/medic.tar.gz`.

7 Acknowledgments

Jared Saia, Nick Kushmerick, and Marc Friedman contributed to our implementation and testing framework. David Smith made many insightful observations that led to a major reformulation of our encoding space. Bart Selman, David McAllester, and Henry Kautz engaged in helpful discussions and kindly provided their Satplan code. Jimi Crawford provided Tableau code.

References

[Blum and Furst, 1995] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proc. 14th Int. Joint Conf. on AI*, pages 1636–1642, 1995.

[Cook and Mitchell, 1997] S. Cook and D. Mitchell. Finding hard instances of the satisfiability problem: A survey. *Proceedings of the DIMACS Workshop on Satisfiability Problems*, To Appear, 1997.

[Crawford and Auton, 1993] J. Crawford and L. Auton. Experimental results on the cross-over point in satisfiability problems. In *Proc. 11th Nat. Conf. on AI*, pages 21–27, 1993.

[Green, 1969] C. Green. Application of theorem proving to problem solving. In *Proc. 1st Int. Joint Conf. on AI*, pages 219–239, 1969.

[Haas, 1987] A. Haas. The case for domain-specific frame axioms. In *The Frame Problem in Artificial Intellegence, Proceedings of the 1987 Workshop*. Morgan Kaufmann, 1987.

[Kautz and Selman, 1992] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. 10th Eur. Conf. on AI*, pages 359–363, Vienna, Austria, 1992. Wiley.

[Kautz and Selman, 1996] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. 13th Nat. Conf. on AI*, pages 1194–1201, 1996.

[Kautz *et al.*, 1996] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proc. 5th Int. Conf. on Principles of Knowledge Representation and Reasoning*, 1996.

[McAllester *et al.*, 1997] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proc. 14th Nat. Conf. on AI*, Providence, Rhode Island, July 1997.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[Penberthy and Weld, 1992] J.S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 103–114, October 1992. See also `http://www.cs.washington.edu/research/projects/ai/www/ucpop.html`.

[Selman *et al.*, 1996] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:521–532, 1996.

[Selman *et al.*, 1997] Bart Selman, Henry Kautz, and David McAllester. Computational challenges in propositional reasoning and search. In *Proc. 15th Int. Joint Conf. on AI*, 1997.

[Van Gelder and Tsuji, 1996] A. Van Gelder and Y. K. Tsuji. Satisfiability testing with more reasoning and less guessing. In D. S. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge.*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.