Planning Via Model Checking: Some Experimental Results

Daniela Berardi, Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
fax: +39 6 85300849
{berardi,degiacomo}@dis.uniroma1.it

Abstract

Model checking is a widely used technique in verification of dynamic systems. Recently several papers have shown that model checking can be used to do planning. In the present paper we report the results of a set of experiments on using two well-known model checkers, Spin and SMV, to do traditional planning (i.e, planning in a complete information setting for reaching a state where a certain goal is satisfied). In the experiments we have compared the performances of such model checkers with state-of-art planners, IPP, FF, and TLPLAN, on problems used in AIPS'98 and AIPS'00 competitions.

Keywords: model-theoretic approaches to planning, planning for temporally extended goals, experimental results

1 Introduction

Model checking is a widely used technique in verification of dynamic systems [8, 16]. Such a technique is based on building the transition graph associated to a dynamic system and verifying on such a transition graph dynamic properties expressing acceptable runs. Sophisticated technologies have been developed to deal efficiently with very large transition systems. As a result model checking is having an increasingly success in the industry especially for hardware verification.

Recently several paper have shown that model checking can be used to do planning [6, 12, 9, 1]. The interest on planning via model checking is especially motivated by the need to extend traditional planning to planning for temporally extended goals [2, 3]. Indeed, model checking can be naturally used to generate plans that are sequences of actions satisfying very general dynamic properties. In addition, model checking can also be used to generate infinite plans, i.e., plans that involve non terminating loops [9].

Another source of interest in planning via model checking is the set of technologies developed to deal efficiently with large search spaces, which are crucial in a real world setting. In the present paper we focus exactly on these. In particular we report the results of a set of experiments that we have done using two well-known model checkers, Spin [14] and SMV [19], to do traditional planning (i.e, planning in a complete information setting for reaching a state where a certain goal is satisfied). In the experiments we have compared the performances of such model checkers with state-of-art planners on problems used in AIPS'98 and AIPS'00 competitions. The planners we have used for the comparison are IPP [17], which was one of the best performers in AIPS'98 competition, FF [13], which was among the best performers in AIPS'00, and TLPLAN [3], which accepts temporally extended goals used as control knowledge to prune that search space.

The rest of paper is organized as follows. In Section 2, we briefly describe model checking and how it can be used to do planning. In Section 3 we introduce the two model checkers and the three planners used in the experiments and the experiments themselves. In Section 4 we report the results of the main experiments. In Section 5 we consider the impact of adding control knowledge, encoded as action preconditions and as temporally extended goals. In Section 6 we draw some conclusion.

2 Planning via model checking

Model checking [8, 16] is an automated technique for verifying finite state systems. The process of determining the correctness of specifications, design and products is growing in importance, since systems are more and more complex and errors are become more and more unacceptable. An important feature of model checking is that it can be performed automatically on finite states systems. On such systems, model checking is typically implemented by an exhaustive search of the state space of the system, to determine if some property is true or not. Given sufficient resources, the procedure will always terminate with a yes/no answer. Moreover, it can be implemented by algorithms [21, 11] with reasonable efficiency, which can be run on moderate-sized machine. Model checking is based on the following ideas:

1. The system is modeled in a specification language accepted by the model checker. Through this language one can describe a transition system, i.e., a transition graph that describes in terms of states and transitions between states (caused by actions) all possible behaviors of the modeled system.

2. The correctness claim, i.e., the property of the system that we want to verify, is expressed as a temporal logic formula that places certain constraints on the behaviors of the system, i.e., on the (possibly infinite) sequences of states the system may traverse in its execution. In particular, in this paper we concentrate on system properties expressed in linear time logic (LTL) [22].

3. The model checker verifies the claim wrt all possible behaviors of the system.

If the model checker finds a behavior that falsify the claim, it returns a counterexample, that is a sequence of states from the initial situation, demonstrating why the claim isn't verified. The main problem the model checkers have to face, is states explosion: usually the transition system that has to be built to verify a claim is exponential in the specification. Hence model checkers implement sophisticated techniques to tackle such a problem.

How can model checking be applied to planning? The idea is very simple. We specify the planning domain in the language accepted by the model checker and the goal as an LTL formula. This formula expresses that eventually a state of the system, where the goal is satisfied, is reached. Then, we give to the model checker the negation of the formula as the correctness claim. The model checker tries to falsify the claim, and if it succeeds, it returns a sequence of states that constitutes a counterexample. Now, in such a sequence, the goal is eventually satisfied. So from such a sequence we can extract the actions that form a plan for the goal.

3 Experiments

We have used for our experiments the model checkers Spin [14] and SMV [19]. These model checkers are widely used in industry to formally verify the correct behavior of synchronous and asynchronous process systems. They both accept a correctness claim specified by an LTL formula[1]. Spin uses a depth-first search algorithm to search for the counterexample, whereas SMV a breadth-first algorithm. Both Spin and SMV implement suitable techniques to solve the states explosion problem.

We have compared the model checkers Spin (precompiled version 3.4.6) and SMV (precompiled version 08-08-00p3) with the planners FF [13] (v2.2), IPP [17] (v3.3) and TLPLAN [3]. IPP and FF are among the planners that scored the best results at AIPS'98 and AIPS'00 competitions, respectively. TLPLAN is a forward-chaining planner that accepts temporally extended goals. These can be used to prune the search space by expressing domain dependent search control knowledge. Since for TLPLAN we can choose the search strategy, we chose the depth first strategy that is faster though it returns plans that are not optimal wrt the length.

[1]In fact SMV accepts CTL formulas, however it is able to translate LTL formulas into CTL using the results in [7].

We have used a Pentium III 1000Mhz, 512Mb RAM and O.S. Windows 2000. To run FF we used the Unix simulator Cygwin-b20.

We have compared the systems on six problems: m10-simple world, m10-full world, schedule world, blocks world, gripper world and briefcase world[2]. The first four were proposed in AIPS'00 competition[3], the fifth in AIPS'98 competition[4] and the last one, proposed by Pednault [20], is often reported in literature[5]. For each of the above problem we have measured the maximum size of the instances that could be solved, the time needed to solve it, and the length (i.e, the number of actions) of the plan returned.

We mostly have used default values for the various parameters of the systems. When different values are used we make it explicit in the text.

4 Results

The first set of experiments are tailored to show how model checkers compete on planning scenarios as drawn directly from planning competitions[6]. We report the results obtained in tables. Each table contains the name of the problem instances, their size in increasing order and, for each system, the seconds it required to find a solution and the number of actions; these two values are marked resp. by an "s" and an "a". To indicate why the planner or the model checker couldn't solve the problem instance, we introduce the following symbols:

- $\not m$ meaning that the system ran out of memory;
- $\not r$ meaning that the system exhausted at least one of the available resources but the memory (the depth search limit, the State-vector size limit, etc.);
- $\not t$ meaning that no solution was found after 30 minutes.

The m10-simple problem. An elevator has to be driven from a floor to another, to board and bring passengers to destination. The size of the problem instances is expressed in terms of number of passengers to serve (the number of floors is twice the number of passengers). The problem instances proposed in the original competition have at most size 30. The table in Figure 1 shows the obtained results.

FF and TLPLAN solve all presented instances, but the former runs faster; SMV does much better than IPP, while Spin is slightly worse. In all presented instances, the model checkers always need a longer time to find a solution, compared to the time required by the planners.

[2]The coding of the problems for the various systems is available from the authors on request.

[3]The specifications, used in the experiments, of the problems m10-simple world, m10-full world, schedule world, blocks world can be found at `http://www.cs.toronto.edu/aips2000/`. On that site are also available the official competition results of AIPS'00.

[4]The specification of gripper world, used in the experiments, can be found at `http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`. On that site are also available the official competition results of AIPS'98.

[5]The specification of briefcase world used in the experiments can be found at `http://www.informatik.uni-freiburg.de/~koehler/ipp.html`

[6]In these experiments TLPLAN runs without exploiting search control knowledge.

In. name	In. size	IPP	FF	TLPLAN	SMV	Spin
1.0	1	0.00 s	0.00 s	0.01 s	0.55 s	7.44 s
		4 a	4 a	4 a	4 a	3 a
...	...	...	...	...	...	...
7.0	7	6.55 s	0.02 s	0.34 s	1.63 s	11.13 s
		18 a	19 a	77 a	18 a	31 a
8.0	8	66.15 s	0.04 s	0.58 s	3.00 s	✗
		22	24 a	106 a	22 a	✗
9.0	9	502.55 s	0.04 s	0.90 s	5.00 s	✗
		26 a	27 a	127 a	26 a	✗
10.0	10	✗	0.07 s	1.11 s	14.73 s	✗
		✗	29 a	127 a	27 a	✗
...	...	...	...	...	...	...
19.0	19	✗	0.22 s	39.12 s	837.17 s	✗
		✗	51 a	424 a	48 a	✗
20.0	20	✗	0.39 s	35.59 s	✗	✗
		✗	53 a	368 a	✗	✗
...	...	...	...	...	...	...
30.0	30	✗	1.39 s	1690.56 s	✗	✗
		✗	73 a	565 a	✗	✗

Figure 1: M10-simple problem.

In. name	In. size	IPP	FF	TLPLAN	SMV	Spin
1.0	1	0.00 s	0.03 s	0.00 s	0.72 s	7.43 s
		4 a	4 a	4 a	9 a	5 a
...	...	...	...	...	...	...
4.0	4	73.85 s	0.03 s	0.01 s	3.88 s	13.11 s
		12 a	12 a	20 a	25 a	21 a
5.0	5	✗	0.06 s	0.04 s	8.27 s	✗
		✗	16 a	39 a	33 a	✗
...	...	...	...	...	...	...
9.0	9	✗	0.03 s	0.55 s	388.20 s	✗
		✗	26 a	77 a	47 a	✗
10.0	10	✗	4.37 s	✗	✗	✗
		✗	34 a	✗	✗	✗
...	...	...	...	...	...	...
30.0	30	✗	73.83 s	✗	✗	✗
		✗	92 a	✗	✗	✗

Figure 2: M10-full problem.

The m10-full problem. It's the same as the previous, but with some additional constraints:

- "conflict_A" and "conflict_B" passengers are never on the lift at same time;
- "never_alone" passengers go on board only if there's an "attendant" passenger on the lift;
- "vip" and "going_nonstop" passengers are served before the others;
- some passengers don't have access to some floors;
- "going_up" passengers can't be on the lift when it goes down;
- "going_down" passengers can't be on the lift when it goes up.

The size of the problem instances is expressed in terms of number of passengers to serve (the number of floors is twice the number of passengers). The problem instances proposed in the original competition have at most size 30. The table in Figure 2 shows the obtained results.

FF is the best. As in the previous problem, SMV solves more instances than IPP. Spin solves the same number of instances solved by IPP. The same happens for TLPLAN and SMV. Observe that the solution length found by SMV is more than twice the others because of the single assignment rule [18] that imposes strict limitations on how the constraints imposed by the problem are verified.

The gripper problem. A robot with two hands has to move a given number of balls from room A to room B. The size of the problem instances is expressed in terms of number of balls to be moved. The instances proposed in the original competition have at most 42 balls. However, we went on with the experiments, increasing the size of problem instances, until the model checkers could not tackle them anymore. The obtained results are shown in the table in Figure 3.

The column Spin* indicates the results obtained by a smart coding the problem that uses some predefined Spin constructs to handle communication across channels. FF is again the best. TLPLAN is the worst. SMV can solve more instances than IPP. Spin, when the specifications are coded in a simple way, can handle at most 10 balls, while when they are coded smartly it can do much better. This shows that one can greatly improve the performance of Spin by hacking with the coding of the problem.

We also changed some of the parameters of Spin to improve its capabilities. We denote with "♯" the results obtained setting the search depth limit (variable Maximum Search Depth) to a value of 200,000 (or greater); with "♯♯" we indicate the results obtained setting the search depth limit to a value greater than 10,000 and the variable DVECTORSZ to a value greater than 1076 (see [15] for details). Spin, even with a smart coding of the problem (Spin*), cannot handle more than 250 balls because of the State-vector size limit ([15]), whose value can be changed only rebuilding Spin.

In. name	In. size	IPP	FF	TLPLAN	SMV	Spin	Spin*
prob01	4	0.00 s	0.02 s	0.09 s	0.89 s	7.40 s	7.40 s
		11 a	11 a	183 a	11 a	159 a	11 a
prob02	6	0.15 s	0.02 s	0.71 s	2.39 s	7.43 s	7.40 s
		17 a	17 a	927 a	17 a	727 a	17 a
prob03	8	3.00 s	0.03 s	⊬	6.43 s	12.63 s	7.49 s
		23 a	23 a	⊬	23 a	7489 a	23 a
prob04	10	39.30 s	0.03 s	⊬	16.79 s	48.08♯ s	7.55 s
		29 a	29 a	⊬	29 a	>20,000♯ a	29 a
prob05	12	438.60 s	0.02 s	⊬	43.90 s	⊬	7.40 s
		35 a	35 a	⊬	35 a	⊬	35 a
prob06	14	⊬	0.03 s	⊬	116.73 s	⊬	7.34 s
		⊬	41 a	⊬	41 a	⊬	41 a
...	...	...	...	...	...	...	...
prob10	22	⊬	0.05 s	⊬	1527.81 s	⊬	7.43 s
		⊬	65 a	⊬	65 a	⊬	65 a
prob11	24	⊬	0.04 s	⊬	⊬	⊬	7.50 s
		⊬	71 a	⊬	⊬	⊬	71 a
...	...	...	...	...	...	...	...
prob20	42	⊬	0.13 s	⊬	⊬	⊬	7.48 s
		⊬	125 a	⊬	⊬	⊬	125 a
...	...	...	...	...	...	...	...
prob59	120	⊬	0.46 s	⊬	⊬	⊬	7.48 s
		⊬	359 a	⊬	⊬	⊬	359 a
prob60	122	⊬	0.86 s	⊬	⊬	⊬	7.56♯♯ s
		⊬	365 a	⊬	⊬	⊬	365♯♯ a
...	...	...	...	...	...	...	...
prob124	250	⊬	3.36 s	⊬	⊬	⊬	7.51♯♯ s
		⊬	749 a	⊬	⊬	⊬	749♯♯ a
prob125	251	⊬	3.66 s	⊬	⊬	⊬	⊬
		⊬	753 a	⊬	⊬	⊬	⊬
...	...	...	...	...	...	...	...
prob334	666	⊬	48.08 s	⊬	⊬	⊬	⊬
		⊬	1997 a	⊬	⊬	⊬	⊬

Figure 3: Gripper problem.

In. name	In. size	IPP	FF	TLPLAN	SMV	Spin	Spin*
2.0	2	0.00 s	0.17 s	∉	0.89 s	∉	7.40 s
		2 a	2 a	∉	4 a	∉	2 a
		1 ts	1 ts	∉	1 ts	∉	2 ts
...	...	...	...	...	...	...	...
4.0	4	0.05 s	0.07 s	∉	24.57 s	∉	7.39 s
		4(+1) a	4(+1) a	∉	11(+1) a	∉	4 a
		2 ts	2 ts	∉	2 ts	∉	2 ts
5.0	5	0.10 s	0.07 s	∉	∉	∉	7.57 s
		4(+1) a	4(+1) a	∉	∉	∉	5 a
		2 ts	2 ts	∉	∉	∉	3 ts
6.0	6	1.75 s	0.09 s	∉	∉	∉	32.69 s
		6(+1) a	7(+4) a	∉	∉	∉	11 a
		2 ts	5 ts	∉	∉	∉	9 ts
7.0	7	2.80 s	0.10 s	∉	∉	∉	100.95 ♯♯ s
		6(+1) a	6(+1) a	∉	∉	∉	7♯♯ a
		2 ts	2 ts	∉	∉	∉	4♯♯ ts
8.0	8	∉	0.41 s	∉	∉	∉	∉
		∉	9(+3) a	∉	∉	∉	∉
		∉	4 ts	∉	∉	∉	∉
...	...	...	...	...	...	...	...
51.0	51	∉	12.37 s	∉	∉	∉	∉
		∉	51(+24) a	∉	∉	∉	∉
		∉	25 ts	∉	∉	∉	∉

Figure 4: Schedule problem.

Observe that by changing some of the parameters we could also improve the capabilities of FF. However, to go over 666 balls, we have to rebuilt it, setting the variable MAX_PLAN_LENGTH to a value greater than 2000. To go over 796 balls we need to rebuilt it, setting also the variable MAX_STATE to a higher value than it is.

The schedule problem. A collection of parts has to be processed by certain machines; goals are mostly non-interacting, but they compete for resources (time on machines) and on the same part different goals clobber other goals. This problem implements the concepts of concurrency and parallelism of actions: this situation is modeled by considering a time-stamp, a variable whose value is incremented each time parts release the machines. The size of the problem instances is expressed in terms of number of parts to be processed. The instances proposed in the original competition have at most size 51. The obtained results are shown in the table in Figure 4, where "ts" marks the number of time-stamps required. The number of actions includes only the actions to process parts and not the ones to increment the time-stamp: the latter are reported in parentheses. Only Spin has a construct to handle concurrency, so we calculate by hand the value of its time-stamp. Again, the column Spin* indicates the results obtained by coding the problem in smart way.

We can see that without hacking Spin cannot solve any problem instance. With a smart

In. name	In. size		IPP	FF	TLPLAN	SMV	Spin
	obj.	loc.					
t1	1	2	0.00 s	0.01 s	0.00 s	0.48 s	7.40 s
			3 a	3 a	3 a	3 a	3 a
...	...	...	...	...	...	...	...
4a	4	5	0.25 s	0.03 s	0.00 s	1.32 s	7.55 s
			9 a	9 a	44 a	9 a	128 a
5	5	6	0.40 s	0.06 s	♯	7.47 s	♯
			17 a	19 a	♯	17 a	♯
...	...	...	...	...	...	...	...
13a	6	7	1148.65 s	0.02 s	♯	76.76 s	♯
			14 a	13 a	♯	13 a	♯
t7	7	8	♯	0.06 s	♯	287.99 s	♯
			♯	15 a	♯	15 a	♯
t8	8	9	♯	0.04 s	♯	♯	♯
			♯	17 a	♯	♯	♯
...	...	...	...	...	...	...	...
t10	10	11	♯	0.16 s	♯	♯	♯
			♯	21 a	♯	♯	♯

Figure 5: Briefcase problem.

coding Spin can solve the same instances as IPP. TLPLAN can't solve any problem instance[7]. SMV can handle at most 4 parts. FF does much better than all the other systems. Finally, we changed some of the parameters of Spin to deal with bigger instances: we indicate with "♯♯" the results obtained using a Supertrace/Bitstate verification (see [15]).

The briefcase problem. A briefcase has to be moved between different locations, to bring objects in their goal locations; objects can be put in and taken out of it, and when they're in the briefcase, they are moved with it. The size of the problem instances is expressed in terms of number of objects and number of locations, reported resp. in columns "obj." and "loc.". The problem instances considered here have at most 10 objects and 11 locations. The obtained results are shown in Figure 5. FF is still the best. SMV and IPP have similar performances. The same happens for Spin and TLPLAN.

The blocks problem. Stack a set of blocks, to reach a given configuration and moving only one block at a time. The size of the problem instances is expressed in terms of number of blocks to be handled. The instances proposed in the original competition have at most 50 blocks. The table in Figure 6 shows the obtained results. "♯" denotes that the variable Maximum Search Depth has been set to 32,000, whereas "♯♯" indicates that it has been set to 1,000,000. FF is the best. Spin and SMV can handle the same number of blocks and they behave worse than IPP. TLPLAN is the worst.

[7] However, we have to remind that TLPLAN has been built to use a control knowledge to guide the search of a plan.

In. name	In. size	IPP	FF	TLPLAN	SMV	Spin
4.0	4	0.05 s	0.02 s	0.25 s	1.00 s	7.33 s
		6 a	6 a	12	6 a	12 a
5.0	5	0.00 s	0.02 s	0.09 s	4.38 s	7.47 s
		12 a	12 a	296 a	12 a	302 a
6.0	6	0.05 s	0.02 s	↯	27.87 s	7.84♯ s
		12 a	20 a	↯	12 a	1760♯ a
7.0	7	0.05 s	0.01 s	↯	484.73 s	27.53♯♯ s
		20 a	20 a	↯	20 a	> 10,000♯♯ a
8.0	8	0.15 s	0.03 s	↯	↯m	↯
		18 a	18 a	↯	↯m	↯
...	...	...	...	...	...	...
12.0	12	3.70 s	0.38 s	↯	↯m	↯
		34 a	44 a	↯	↯m	↯
...	...	...	...	...	...	...
13.0	13	↯m	0.04 s	↯	↯m	↯
		↯m	44 a	↯	↯m	↯
...	...	...	...	...	...	...
50.0	50	↯m	1.85 s	↯	↯m	↯
		↯m	172 a	↯	↯m	↯

Figure 6: Blocks problem.

5 Adding control knowledge

The second set of experiments were designed to show whether one could take advantage of the additional expressive power provided by LTL for expressing control knowledge. Control knowledge consists of suitable constraints on state transitions and thus can (at least in principle) be used to reduce the state space explored during planning. Obviously verifying such constraints adds an overhead in choosing the state to explore next. Hence, control knowledge is effective when the reduction of the state space is prevailing over the overhead introduced in processing each state.

Control knowledge can either be expressed in a declarative way through a temporally extended goal, or in a "procedural way" changing the system itself by modifying the preconditions of actions. The difference between these two approaches is that while the former requires the system to extract from constraints on transitions the corresponding conditions on the next state to explore, in the latter such conditions are already explicited in the preconditions of actions.

Traditional planning systems do not allow for expressing temporally extended goals. So for these systems phrasing control knowledge through action preconditions is the only choice. TLPLAN has a special construct for the specification of control knowledge as a temporally extended goal. For the model checkers we adopted both approaches, though SMV posed difficulties in expressing certain action preconditions (see below).

We have concentrated on briefcase world and blocks world problems. On those we have implemented the control knowledge in [3].

In. name.	IPP (Int)	FF (Int)	TLPLAN (Goal)	SMV (Int)	SMV (Goal)	Spin (Int)	Spin (Goal)
t1	0.00 s	0.11 s	0.00 s	0.43 s	0.58 s	7.43 s	7.64 s
	3 a	3 a	3 a	3 a	5 a	3 a	3 a
...	...	...	...	...	...	...	...
4a	37.10 s	0.37 s	0.01 s	0.89 s	50.70 s	7.30 s	7.52 s
	9 a	9 a	15 a	9 a	19 a	9 a	9 a
5	∉	2.45 s	0.01 s	2.40 s	∉	7.48 s	7.59 s
	∉	22 a	23 a	17 a	∉	20 a	20 a
...	...	...	...	...	...	...	...
t7	∉	30.73 s	0.02 s	96.40 s	∉	7.70 s	7.58 s
	∉	15 a	27 a	15 a	∉	15 a	15 a
t8	∉	∉	0.03 s	∉	∉	7.68 s	7.46 s
	∉	∉	31 a	∉	∉	17 a	17 a
...	...	...	...	...	...	...	...
t10	∉	∉	0.05 s	∉	∉	7.91 s	7.77 s
	∉	∉	39 a	∉	∉	21 a	21 a

Figure 7: Briefcase problem with control knowledge.

The briefcase problem. The control knowledge for this problem is based on the following natural ideas:

1. Don't move the briefcase from its current location if there is an object that needs to be taken out or put into the briefcase;
2. Don't take an object out of the briefcase if the briefcase is not at the object's goal location;
3. Don't put objects that don't need to be moved into the briefcase;
4. Don't move the briefcase to an irrelevant location, where a location is irrelevant if there is no object to be picked up there, there is no object in the briefcase that needs to be dropped off there, and it is not a goal to move the briefcase to that location.

The obtained results are shown in the table in Figure 7, where:

- (Pre) means that the control knowledge is implemented within the system,
- (Goal) that it's implemented within the goal specification.

Spin and TLPLAN solve all problem instances: the former finds shorter solutions, whereas the latter requires a shorter time. SMV solves the same instances as IPP, if we encode the control knowledge within the system. It solves the same instances as FF, if we encode the control knowledge within the goal. In Spin, we had difficulties in encoding the control knowledge within the goal: the LTL formula, encoding the control knowledge, involves the use of the LTL next operator X, but the precompiled version of Spin doesn't support it. We managed to encode the control knowledge by hacking with Spin predefined constructs to handle communication across channels.

In. name	IPP (Pre)	FF (Pre)	TLPLAN (Goal)	Spin (Goal)	Spin (Pre)	Spin (Pre*)	SMV (Pre*)
4.0	0.10 s	0.12 s	0.00 s	9.99 s	7.87 s	7.20 s	3.29 s
	6 a	6 a	6 a	6 a	6 a	6 a	6 a
5.0	0.25 s	0.03 s	0.00 s	11.50 s	8.53 s	7.29 s	↯
	12 a	12 a	12 a	12 a	12 a	12 a	↯
6.0	0.55 s	0.03 s	0.01 s	↯	9.27 s	7.57 s	↯
	12 a	12 a	18 a	↯	14 a	14 a	↯
...	...	...	...	...	...	...	...
9.0	3.80 s	0.02 s	0.02 s	↯	13.64 s	8.66 s	↯
	30 a	30 a	30 a	↯	30 a	30 a	↯
10.0	6.50 s	0.01 s	0.02 s	↯	↯	9.12 s	↯
	34 a	34 a	34 a	↯	↯	34 a	↯
...	...	...	...	...	...	...	...
15.0	78.15 s	0.06 s	0.09 s	↯	↯	13.99 s	↯
	40 a	40 a	56 a	↯	↯	48 a	↯
16.1	↯m	0.04 s	0.12 s	↯	↯	↯	↯
	↯m	54 a	56 a	↯	↯	↯	↯
...	...	...	...	...	...	...	...
50.0	↯m	0.15 s	3.17 s	↯	↯	↯	↯
	↯m	170 a	184 a	↯	↯	↯	↯

Figure 8: Blocks problem with control knowledge.

The blocks problem. The control knowledge for this problem is based on the classification of towers as good or bad:

ck1: "a good tower can't be destroyed, a bad tower can't grow up with blocks, it isn't useful to pick up a block if the tower where we want to stack it isn't a good tower".

For the two model checkers we also report the results obtained adding to the system, instead of the previous control knowledge, the simpler condition:

ck2: "stack block x on block y if y is on its goal-block and y is x goal-block, where goal-block is the block where the considered block is on, in the goal situation."

In fact, we could encode ck1 in SMV neither within the system, nor within the goal, in a simple way, because of the single assignment rule [18]. We also had difficulties in encoding ck1 in Spin within the goal because of the LTL next operator X that the precompiled version of Spin doesn't support. However, we could encode ck1 by doing the same hacking as for the briefcase problem. The obtained results are shown in Figure 8, where:

- (Pre) means that the control knowledge ck1 is implemented within the system,
- (Goal) that ck1 is implemented within the goal specification,
- (Pre*) that we implemented the control knowledge ck2 within the system.

FF and TLPLAN are the best. Spin can solve the same instances as IPP if we encode control knowledge ck2; if we encode ck1, instead, Spin solves less instances. SMV can handle at most 4 blocks encoding control knowledge ck2.

6 Conclusions

The results of the experiments shows that the performances of the two model checkers, Spin and SMV, are comparable to that of IPP. Instead FF performs much better that both. In other words Spin and SMV used as planners are competitive with the best performing planners at the AIPS'98 competition but not with those at AIPS'00. Spin can indeed improve its performance by exploiting additional control knowledge. Instead SMV appears to be less responsive to control knowledge. With control knowledge, TLPLAN becomes the best performer. It has to be pointed out that Spin is quite sensible to how the problem specification is coded. Often by hacking with such coding one can greatly improve its performance.

We observe that there is a lot of room for improvement in doing planning using Spin and SMV, and more generally using model checkers. Our experiments show that both Spin and SMV are not fine tuned to do planning. Indeed, as shown, by changing various built in parameters we can get better performances already. Obviously one could do even better by exploiting the technologies at the base of such model checkers to devise new planning systems that take advantage of them. First experiments in this directions are quite encouraging see for example [10].

There are several other systems we would have liked to do our experiments on: among them, the planner PADOK [4], which accepts LTL goals as Spin, SMV, and TLPLAN, and also the model checker NuSMV [5], a recent re-implementation of SMV.

References

[1] M. R. A. Cimatti. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[2] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. of Mathematics and Artificial Intelligence*, 22:5–27, 1998.

[3] F. Bacchus and F. Kabanza. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence*, 16:123 – 191, 2000.

[4] M. Cialdea Mayer, C. Limongelli, A. Orlandini, and G. Balestrieri. A Planner Fully Based on Linear Time Logic. In *Proceeding of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, pages 347 – 354. AAAI Press, 2000.

[5] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[6] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI'98)*, pages 875–881, 1998.

[7] E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another Look at LTL Model Checking. In *Proceedings of the 6th International Conference on Computer Aided Verification (CAV-94)*, pages 415–427, 1994.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, 1999.

[9] G. De Giacomo and M. Y. Vardi. Automata-Theoretic Approach to Planning for Temporally Extended Goals. In *Proceedings of the 5th European Conference on Planning (ECP-99)* LNAI 1809, pages 226–238. Springer, 1999.

[10] S. Edelkamp and M. Helmert. The Model Checking Integrated Planning System (MIPS). *AI-Magazine*, 2001. To appear.

[11] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings 13th Symposium on Protocol Specification, Testing and Verification*, pages 3 – 18, 1995.

[12] F. Giunchiglia and P. Traverso. Planning as Model Checking. In *Proceedings of the 5th European Conference on Planning (ECP-99)* LNAI 1809, pages 1–20. Springer, 1999.

[13] J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research.* To appear. (FF is available at http://www.informatik.uni-freiburg.de/~hoffmann/ff.html).

[14] G. J. Holzmann. The Model Checker Spin. *IEEE Transactions on Software Engineerings*, 23(5):279 – 295, 1997.

[15] G. J. Holzmann et al. Spin Online References. (Spin is available at http://netlib.bell-labs.com/netlib/spin/index.html or http://netlib.bell-labs.com/netlib/spin/whatispin.html).

[16] J.-P. Katoen. *Concepts, Algorithms, and Tools for Model Checking.* Friedrich-Alexander-Universität Erlangen Nürnberg, 1999. Lecture notes of the Course "Mechanised Validation of Parallel Systems" – Semester 1998/1999.

[17] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending Planning Graphs to an ADL Subset. In *Proceedings of the 4th European Conference in Planning (ECP-97)* LNCS 1348, pages 273–285. Springer, 1997. (IPP is available at http://www.informatik.uni-freiburg.de/~koehler/ipp.html).

[18] K. L. McMillan. *The SMV Model Checking System.* Cadence Berkeley Labs. (SMV is available at http://www-cad.eecs.berkeley.edu/~kenmcmill/smv/).

[19] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[20] E. Pednault. ADL: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 324–332, 1989.

[21] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146 – 160, 1972.

[22] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moeller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, LNCS 1043, pages 238 – 266. Springer-Verlag, 1996.