4. Case $g = g_1 \, \mathbf{U} \, g_2$. ($\Rightarrow$) Assume that $\pi_j \models g_1 \, \mathbf{U} \, g_2$, then for some $l \geq j$, $\pi_l \models g_2$ and for all $j \leq i < l$, $\pi_i \models g_1$. By the induction hypothesis, $s_l \in sat(g_2)$ and therefore $s_l \in sat(g_1 \, \mathbf{U} \, g_2)$. By the definition of R_T, it follows that $s_{l-1} \in sat(\mathbf{X}(g_1 \, \mathbf{U} \, g_2))$. But $\pi_{l-1} \models g_1$, so, by induction $s_{l-1} \in sat(g_1)$ and therefore $s_{l-1} \in sat(g_1 \, \mathbf{U} \, g_2)$. By induction on $(l - j)$ we eventually get $s_j \in sat(g_1 \, \mathbf{U} \, g_2)$.

   ($\Leftarrow$) Suppose that $s_j \in sat(g_1 \, \mathbf{U} \, g_2)$ and $\pi_j \not\models g_1 \, \mathbf{U} \, g_2$. By Lemma 7, for all $k \geq j$, $s_k \in sat(g_1 \, \mathbf{U} \, g_2)$ and $\pi_k \not\models g_1 \, \mathbf{U} \, g_2$. This implies that $\pi_k \not\models g_2$, and thus $s_k \notin sat(g_2)$ from the induction hypothesis. Consequently $s_k \in sat(g_1 \, \mathbf{U} \, g_2)$ and $s_k \notin sat(g_2)$ for all $k \geq j$. This leads to a contradiction, because $\pi \models \mathbf{G}\, True$ guarantees that there are infinitely many states s_k such that $s_k \in sat(\neg(g_1 \, \mathbf{U} \, g_2) \vee g_2)$. Therefore if $s_j \in sat(g_1 \, \mathbf{U} \, g_2)$, then $\pi_j \models g_1 \, \mathbf{U} \, g_2$. $\Box$

Proof of Theorem 2. ($\Rightarrow$) Since $M, s'_0 \models \mathbf{E}\, f$, then $\exists \pi' \models f$. By Theorem 1 and Lemma 5, we can prove, for π in T, $\pi \models \mathbf{G}\, True$ and $label(\pi) = label(\pi') \mid_{AP_f}$. By Lemma 6, there is a path π'' in P such that $label(\pi'') = label(\pi)$. Since $label(\pi) = label(\pi') \mid_{AP_f}$ and $\pi' \models f$, we can see $\pi \models f$. Also since $\pi \models \mathbf{G}\, True$, by Lemma 8 $s_0 \in sat(f)$. Thus $(s_0, s'_0) \in sat(f)$. Since $label(\pi) = label(\pi'')$ and $\pi \models \mathbf{G}\, True$, it is clear that $\pi'' \models \mathbf{G}\, True$. Therefore $P, (s_0, s'_0) \models \mathbf{EG}\, True$.

($\Leftarrow$) Since $(s_0, s'_0) \in sat(f)$ and $P, (s_0, s'_0) \models \mathbf{EG}\, True$, then $\exists \pi'' \models \mathbf{G}\, True$. By Lemma 6, there exist paths $\pi \in T$ and $\pi' \in M$ such that $label(\pi'') = label(\pi) = label(\pi') \mid_{AP_f}$. Since $\pi'' \models \mathbf{G}\, True$ and $label(\pi) = label(\pi'')$, we can see $\pi \models \mathbf{G}\, True$. Since $(s_0, s'_0) \in sat(f)$, $s_0 \in sat(f)$. From Lemma 8, $\pi \models f$. Since $label(\pi) = label(\pi') \mid_{AP_f}$, $\pi' \models f$. Therefore $M, s'_0 \models \mathbf{E}\, f$. $\Box$

Proof. Clearly, for all i, $s_i \in S_T$. By Lemma 3 and the definition of $\mathbf{X}$, it is easy to see the following relation: $s_i \in sat(\mathbf{X}\, g)$ if and only if $M, \pi'_i \models \mathbf{X}\, g$ if and only if $M, \pi'_{i+1} \models g$ if and only if $s_{i+1} \in sat(g)$. By the definition of R_T, if $s_i \in sat(\mathbf{X}\, g) \Leftrightarrow s_{i+1} \in sat(g)$, then $(s_i, s_{i+1}) \in R_T$. Therefore $\pi = s_0 s_1 \ldots$ is a path in T. □

Proof of Theorem 1. Suppose that, for a path π' in M, $\pi' \models f$. By Lemma 4, we can find a path $\pi = s_0 s_1 \ldots$ in T. By Lemma 3, $s_0 \in sat(f)$. By the definition of s_i, $L(s'_i)\mid_{AP_f} = L_T(s_i)$, and thus $label(\pi')\mid_{AP_f} = label(\pi)$. This leads to Theorem 1. □

Theorem 2 $M, \sigma' \models \mathbf{E} f$ if and only if there is a state σ in T such that $(\sigma, \sigma') \in sat(f)$ and $P, (\sigma, \sigma') \models$ $\mathbf{EG}\, True$ under the fairness constraints given in (1).

In order to prove this theorem, we need the following three lemmas.

Lemma 5 *Given $\pi = s_0 s_1 \ldots$ where s_i is defined as above, then $\pi \models \mathbf{G}\, True$ under the the fairness constraints given in (1).*

Proof. In order to show that $\pi \models \mathbf{G}\, True$ under the fairness constraints, we need to prove that, for every subformula $g\, \mathbf{U}\, h$ of f, there are infinitely many states s_i on π such that $s_i \in sat(\neg(g\, \mathbf{U}\, h) \vee h)$. Suppose not, then there exists i_0 such that, for all $i \geq i_0$, $s_i \notin sat(\neg(g\, \mathbf{U}\, h) \vee h)$. Thus $s_i \in sat(g\, \mathbf{U}\, h)$ and $s_i \notin sat(h)$. By Lemma 3, for all $i \geq i_0, \pi'_i \models g\, \mathbf{U}\, h$ and $\pi'_i \not\models h$. Since $\pi'_i \models g\, \mathbf{U}\, h$ means $\pi'_j \models h$ for some $j \geq i$, this leads to a contradiction. □

It is easy to see the next lemma.

Lemma 6 *$\pi'' = (s_0, s'_0)(s_1, s'_1) \cdots$ is a path in P with $L_P((s_i, s'_i)) = L_T(s_i)$ for all $i \geq 0$ if and only if there exist a path $\pi = s_0 s_1 \cdots$ in T, and a path $\pi' = s'_0 s'_1 \cdots$ in M with $L_T(s_i) = L_M(s_i)\mid_{AP_f}$ for all $i \geq 0$.*

Lemma 7 *Asuume that, for all $k \geq j$, $s_k \in sat(g_1) \Leftrightarrow \pi_k \models g_1$ and $s_k \in sat(g_2) \Leftrightarrow \pi_k \models g_2$. If $\pi_j \not\models g_1 \mathbf{U} g_2$ and $s_j \in sat(g_1\, \mathbf{U}\, g_2)$, then, for all $k \geq j$, $\pi_k \not\models g_1\, \mathbf{U}\, g_2$ and $s_k \in sat(g_1\, \mathbf{U}\, g_2)$.*

Proof. First we prove that, if $s_j \in sat(g_1 \mathbf{U} g_2)$ and $\pi_j \not\models g_1 \mathbf{U} g_2$, then $s_{j+1} \in sat(g_1 \mathbf{U} g_2)$ and $\pi_{j+1} \not\models g_1 \mathbf{U} g_2$. From the definition of sat, $sat(g_1\, \mathbf{U}\, g_2)$ implies $s_j \in sat(g_2)$ or ($s_j \in sat(g_1)$ and $s_j \in sat(\mathbf{X}(g_1\, \mathbf{U}\, g_2))$). From the assumptions and the definition of R_T, it follows that:

$$\pi_j \models g_2 \text{ or } (\pi_j \models g_1 \text{ and } s_{j+1} \in sat(g_1\, \mathbf{U}\, g_2)). \tag{2}$$

Since $\pi_j \not\models g_1\, \mathbf{U}\, g_2$ implies $\pi_j \not\models g_2$, (2) leads to the following:

$$\pi_j \models g_1 \text{ and } s_{j+1} \in sat(g_1\, \mathbf{U}\, g_2). \tag{3}$$

Since $\pi_j \models g_1$ from (3) and $\pi_j \not\models g_1\, \mathbf{U}\, g_2$ from the assumption, we can also get $\pi_{j+1} \not\models g_1\, \mathbf{U}\, g_2$. Similarly we can get, for all $k = j+2, j+3, j+4 \ldots$, $s_k \in sat(g_1\, \mathbf{U}\, g_2)$ and $\pi_k \not\models g_1\, \mathbf{U}\, g_2$. □

Lemma 8 *Let $\pi \models \mathbf{G}\, True$ under the fairness constraints, then $T, \pi \models f$ if and only if $s_0 \in sat(f)$.*

Proof. By induction on the structure of the formula, we prove, for each $g \in sub(f) \cup el(f)$, $\forall j : T, \pi_j \models g$ if and only if $s_j \in sat(g)$.

1. Case $g = p \in AP$. By the definition of s_j and the definition of sat, it is easy to see the following relation: $\pi_j \models p$ if and only if $p \in L_T(s_j)$ if and only if $p \in s_j$ if and only if $s_j \in sat(p)$.

2. Case $g = \neg g_1$ and $g = g_1 \vee g_2$. By the induction hyposthesis and the defition of $\neg$ and $\vee$, it is easy to prove these cases.

3. Case $g = \mathbf{X}\, g_1$. By the definition of R_T and the induction hypothesis, we can see the following relation: $s_j \in sat(\mathbf{X}\, g_1)$ if and only if $s_{j+1} \in sat(g)$ if and only if $\pi_{j+1} \models g$ if and only if $\pi_j \models \mathbf{X}\, g$.

and Perspectives (Proceedings of REX School), Lecture Notes in Computer Science, pages 124–175. Springer-Verlag, 1993.

[11] O. Coudert, J. C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In R. P. Kurshan and E. M. Clarke, editors, *Proceedings of the 1990 Workshop on Computer-Aided Verification*, June 1990.

[12] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: On branching time versus linear time. *Journal of the ACM*, 33:151–178, 1986.

[13] E.A. Emerson and Chin Laung Lei. Modalities for model checking: Branching time strikes back. *Twelfth Symposium on Principles of Programming Languages, New Orleans, La.*, January 1985.

[14] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, January 1985.

[15] A. J. Martin. The design of a self-timed circuit for distributed mutual exclusion. In H. Fuchs, editor, *Proceedings of the 1985 Chapel Hill Conference on Very Large Scale Integration*, 1985.

[16] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic Publishers, 1993.

[17] A. P. Sistla and E.M. Clarke. Complexity of propositional temporal logics. *Journal of the ACM*, 32(3):733–749, July 1986.

[18] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, June 1986.

Appendix

We prove Theorem 1 and Theorem 2 of Section 5.

Theorem 1 Let T be the tableau for the path formula f. Then, for every Kripke structure M and every path π' of M, if $M, \pi' \models f$ then there is a path π in T that starts in a state in $sat(f)$, such that $label(\pi') \mid_{AP_f} = label(\pi)$.

In order to prove this theorem, we need the following two lemmas. In the remainder of this section, $\pi' = s'_0 s'_1 \ldots$ represents a path in M. We denote the suffix of π' starting from the state s'_i as π'_i i.e., $\pi'_i = s'_i s'_{i+1} \ldots$. For the path π'_i, we define $s_i = \{\psi | \psi \in el(f) \text{ and } M, \pi'_i \models \psi\}$. Note that s_i is a state in T.

Lemma 3 *For all $g \in sub(f) \cup el(f)$, $M, \pi'_i \models g$ if and only if $s_i \in sat(g)$.*

Proof. The proof proceeds by induction on the structure of the formula.

1. Case $g \in el(f)$. By the definition of s_i, it is easy to see that $M, \pi'_i \models g$ if and only if $g \in s_i$. By the definition of sat, $g \in s_i$ if and only if $s_i \in sat(g)$.

2. Case $g = \neg g_1$ and $g = g_1 \vee g_2$. By the induction hypothesis and the definition of sat, it is easy to prove these cases.

3. Case $g = g_1 \mathbf{U} g_2$. By the definition of $\mathbf{U}$, $M, \pi'_i \models g_1 \mathbf{U} g_2$ if and only if $M, \pi'_i \models g_2$ or $(M, \pi'_i \models g_1$ and $M, \pi'_i \models \mathbf{X}(g_1 \mathbf{U} g_2))$. By the induction hypothesis and the definition of s_i, $M, \pi'_i \models g_2$ or $(M, \pi'_i \models g_1$ and $M, \pi'_i \models \mathbf{X}(g_1 \mathbf{U} g_2))$ if and only if $s_i \in sat(g_2) \vee (s_i \in sat(g_1) \wedge s_i \in sat(\mathbf{X}(g_1 \mathbf{U} g_2)))$. By the definition of sat, $s_i \in sat(g_2) \vee (s_i \in sat(g_1) \wedge s_i \in sat(\mathbf{X}(g_1 \mathbf{U} g_2)))$ if and only if $s_i \in sat(g_1 \mathbf{U} g_2)$. □

Lemma 4 *Given $\pi' = s'_0 s'_1 \cdots$ and s_i as above, then $\pi = s_0 s_1 \ldots$ is a path in T.*

#cell	#nodes	#time(sec)	trans.	#reachable states
3	1994	0.2	149	6144
4	2929	0.2	211	32768
5	3959	0.23	273	163840
6	5039	0.3	335	786432
7	6164	0.31	397	3.67002e+06
8	7334	0.36	459	1.67772e+07
9	8549	0.43	521	7.54975e+07
10	9809	0.43	583	3.35544e+08
11	10050	0.66	645	1.4764e+09
12	10154	0.68	707	6.44245e+09

Table 7: Specification $\mathbf{AG}(\mathbf{G}\,\neg req_1 \rightarrow \mathbf{F}\,\mathbf{G}\,\neg ack_1)$ for the synchronous arbiter

#cell	#nodes	#time(sec)	trans.	#reachable states
3	2769	0.28	139	3072
4	3910	0.41	201	16384
5	5204	0.51	263	81920
6	6638	0.66	325	393216
7	8210	0.81	387	1.83501e+06
8	9919	0.96	449	8.38861e+06
9	10013	1.36	511	3.77487e+07
10	10022	1.48	573	1.67772e+08
11	10048	1.7	635	7.38198e+08
12	10059	2.1	697	3.22123e+09

Table 8: Specification $\mathbf{AG}((\neg req_1 \wedge \mathbf{X}\, req_1) \rightarrow \mathbf{F}\, ack_1)$ for the synchronous arbiter

[5] E. M. Clarke and I. A. Draghicescu. Expressibility results for linear time and branching time logics. In *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354, pages 428–437. Springer-Verlag: Lecture Notes in Computer Science, 1988.

[6] E. M. Clarke, I. A. Draghicescu, and R. P. Kurshan. A unified approach for showing language containment and equivalence between various types of ω-automata. In A. Arnold and N. D. Jones, editors, *Proceedings of the 15th Colloquium on Trees in Algebra and Programming*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1990.

[7] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[9] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Futurebus+ cache coherence protocol. In L. Claesen, editor, *Proceedings of the Eleventh International Symposium on Computer Hardware Description Languages and their Applications*. North-Holland, April 1993.

[10] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Decade of Concurrency – Reflections*

#cell	#nodes	#time(sec)	trans.	#reachable states
3	41511	1340.5	3023	105264
4	78495	6568.6	5002	1.20275e+06
5	133308	23971.3	7005	1.28388e+07
6	200447	73035.4	9008	1.31466e+08
7	278862	204469.5	11011	1.30854e+09

Table 5: Specification $\mathbf{AG}(\mathbf{G}\,\neg req_1 \rightarrow \mathbf{F}\,\mathbf{G}\,\neg ack_1)$ for the DME circuit

#cell	#nodes	#time(sec)	trans.	#reachable states
3	35985	1231.0	3013	52632
4	75807	5942.7	4992	601376
5	124453	22137.5	6995	6.4194e+06
6	190134	70295.0	8998	6.57328e+07
7	274124	186158.6	11001	6.54272e+08

Table 6: Specification $\mathbf{AG}((\neg req_1 \wedge \mathbf{X}\, req_1) \rightarrow \mathbf{F}\, ack_1)$ for the DME circuit

to allow LTL specifications. We evaluated our approach on the DME circuit and the synchronous arbiter. For the safety property and the liveness property that can be described in CTL, the experiments showed that model checking in LTL is as efficient as in CTL. The model checker also succeeded in verifying non-trivial LTL formulas on the circuits. The results that we have obtained suggests that LTL model checking is possible for formulas that are not excessively complicated.

This, however, does not mean that LTL will take the place of CTL in model checking applications. Many other problems, like testing inclusion and equivalence between various types omega-automata [6], can also be reduced to CTL model checking. LTL, on the other hand, does not appear to have this flexibility. Moreover, in many of the applications of model checking to verification, it is important to be able to assert the existence of a path that satisfies some property. For example, *absence of deadlock* might be expressed by the CTL formula $\mathbf{AG}\,\mathbf{EF}\, start$ (Regardless of what state the program enters, there exists a computation leading back to the *start* state). Neither this formula nor its negation can be expressed in LTL [5], so LTL model checking techniques cannot be used to decide whether the formula is true or not.

Idealy, it should be possible to reason about linear-time and branching-time properties in the same logic (say, CTL*). We believe this goal can potentially be realized by extending the techniques discusssed in this paper. Emerson and Lei [13] have shown how to reduce CTL* model checking to LTL model checking. If the transformation outlined in this paper can be extended to incorporate their reduction, then it should be possible to develop a model checker that can handle both types of properties.

References

[1] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[2] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*. IEEE Computer Society Press, June 1990.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

#cell	#nodes		#time(sec)		trans.		#reachable states	
	CTL	LTL	CTL	LTL	CTL	LTL	CTL	LTL
3	384	734	0.08	0.1	80	122	384	768
4	654	1279	0.1	0.1	112	218	2048	4096
5	987	1913	0.11	0.15	144	318	10240	20480
6	1383	2628	0.13	0.18	176	418	49152	98304
7	1842	3424	0.16	0.21	208	518	229376	458752
8	2364	4301	0.16	0.26	240	618	1.04858e+06	2.09715e+06
9	2949	5259	0.16	0.33	272	718	4.71859e+06	9.43718e+06
10	3597	6298	0.21	0.33	304	818	2.09715e+07	4.19430e+07
11	4308	7418	0.21	0.41	336	918	9.22747e+07	1.84549e+08
12	5082	8619	0.31	0.45	368	1018	4.02653e+08	8.05306e+08

Table 3: Safety specification for the synchronous arbiter

#cell	#nodes		#time(sec)		trans.		#reachable states	
	CTL	LTL	CTL	LTL	CTL	LTL	CTL	LTL
3	996	2159	0.10	0.26	80	134	384	1536
4	1531	3137	0.20	0.36	112	196	2048	8192
5	2155	4254	0.38	0.43	144	258	10240	40960
6	2867	5483	0.43	0.48	176	320	49152	196608
7	3667	6820	0.48	0.61	208	382	229376	917504
8	4555	8266	0.53	0.81	240	444	1.04858e+06	4.1943e+06
9	5531	9821	0.71	1.01	272	506	4.71859e+06	1.88744e+07
10	6595	10000	0.83	1.23	304	568	2.09715e+07	8.38861e+07
11	7747	10001	1.00	1.46	336	630	9.22747e+07	3.69099e+08
12	8987	10052	1.16	1.71	368	692	4.02653e+08	1.61061e+09

Table 4: Liveness specification for the synchronous arbiter

- If a request signal raises, then the request is eventually acknowledged.

In this case, the LTL specification has the following form:

$$\mathbf{AG}((\neg req_1 \wedge \mathbf{X}\, req_1) \rightarrow \mathbf{F}\, ack_1)$$

In the experiments, we used the same options for SMV as in the previous experiments. We only checked the two specifications with respect to a single user (user 1). Table 5 and Table 6 show the results for the DME circuit. In the experiments for the first LTL formula, we observe that the amount of the allocated OBDD nodes and run time is quite similar to that for the LTL model checking of the liveness specification. The increase in time and space is less than 10%. In the experiments for the second LTL formula (modified liveness), the time and space that is required is almost the same as for the LTL model checking of the liveness specification. Table 7 and Table 8 summarize the results for the synchronous arbiter. In the experiments for the first LTL formula and the second LTL formula, the amount of time and space that is required is, respectively, 0.5 times and 1.2 times as large as for the LTL model checking of liveness specification. All of these results show the feasibility of LTL model checking for non-trivial formulas.

8 Directions for Future Research

In this paper, we discussed efficient symbolic model checking in LTL. We showed an algorithm for reducing LTL model checking to CTL model checking. Using this algorithm, we extended the SMV model checker

#cell	#nodes		#time(sec)		trans.		#reachable states	
	CTL	LTL	CTL	LTL	CTL	LTL	CTL	LTL
3	11326	11362	17.9	20.5	2778	2781	6579	13158
4	13458	15357	47.5	49.4	4757	4760	75172	150344
5	22321	22348	100.5	104.4	6760	6763	802425	1.60485e+06
6	25869	27318	182.3	193.6	8763	8766	8.2166e+06	1.64332e+07
7	28413	33310	326.4	329.3	10766	10769	8.1784e+07	1.63568e+08
8	44322	44369	509.2	526.3	12769	12772	7.97393e+08	1.59479e+09
9	49702	49755	794.0	794.8	14772	14775	7.65302e+09	1.53060e+10
10	55082	55141	1125.2	1362.7	16775	16778	7.30144e+10	1.46029e+11

Table 1: Safety specification for the DME circuit

#cell	#nodes		#time(sec)		trans.		#reachable states	
	CTL	LTL	CTL	LTL	CTL	LTL	CTL	LTL
3	12721	33940	426.1	1260.5	2778	3004	6579	26316
4	26541	72029	2553.2	6096.7	4757	4983	75172	300688
5	47346	120299	9623.1	21950.1	6760	6986	802425	3.2097e+06
6	92080	183043	36995.3	66502.5	8763	8989	8.2166e+06	3.28664e+07
7	163867	263380	97807.1	191990.0	10766	10992	8.1784e+07	3.27136e+08

Table 2: Liveness specification for the DME circuit

2. (*Liveness*) All requests are eventually acknowledged.

In fact, exactly the same LTL and CTL specifications can be used.

In the experiments using SMV, we used the options to construct single transition relations, and to compute reachable states before model checking. Table 3 shows the experimental results for the safety specification and Table 4 shows the results for the liveness specification. For the safety specification we observe that the number of reachable states for LTL model checking checking is twice as large as for CTL model checking. The number of the allocated OBDD nodes and run time both increase by a factor of 1.5. In the second experiment, the number of the reachable states is four times larger for LTL model checking. The amount of space and time that is required is 1.5–2 times larger.

7.2 Non-Trivial LTL Specifications

In order to examine the feasibility of the LTL model checking, we verified two specifications that are described by non-trivial formulas. There is no proof that these formulas are inexpressible in CTL. It is, however, not obvious how to describe these specifications in CTL, and would be very complicated and unreadable. Again, we used the DME circuit and the synchronous arbiter. For each of these circuits, the gate models are the same as in the previous experiments for the liveness specifications. The first specification that we verify is as follows:

- If a user stops making a request, then the acknowledge signal eventually lowers and stays low forever.

Suppose that the acknowledge signal is high, and the user stops making requests. Typically there are some delays until the circuit lowers the acknowledge signal. Thus, we allow arbitrary finite delays for the circuit to lower the acknowledge signal. In this case, the LTL specification for a user (user 1) has the following form:

$$\mathbf{AG}(\mathbf{G}\,\neg req_1 \rightarrow \mathbf{F}\,\mathbf{G}\,\neg ack_1)$$

The second specification is a modified liveness property:

```
MODULE main  -- simple program

VAR

a: boolean;
b: boolean;

TRANS ( a & !b) -> next(!(a & !b))
TRANS ( a &  b) -> next(a & !b)
TRANS (!a &  b) -> next(!a & b)
TRANS (!a & !b) -> next(!a & b)

VAR

EL_X_a_U_b : boolean;

DEFINE

S_a            := a;
S_b            := b;
S_X_a_U_b      := EL_X_a_U_b;
S_a_U_b        := S_b | (S_a & S_X_a_U_b);
S_NOT_a_U_b    := !S_a_U_b;

TRANS     S_X_a_U_b = next(S_a_U_b)

FAIRNESS       !S_a_U_b | b

SPEC      !(S_NOT_a_U_b & EG true)
```

Figure 7: Translator output for simple SMV program

- A single OBDD is constructed for the transition relation of the circuit.
- The reachable states of the circuit are determined, and evaluation of the CTL operators is restricted to these states.
- At each step in the forward search, the transition relation is restricted to the set of reachable states. The *Restrict* function of Coudert, Madre and Berthet [11] is used for this purpose.

Table 1 summarizes the experimental results for the safety specification, and Table 2 summarizes the results for the liveness specification. The columns show the number of the cells (**#cell**), the maximum number of OBDD nodes used at any given time (**#nodes**), the run time on SPARC station 10 (**time**), the size of the transition relation in OBDD nodes (**trans.**) and the number of the reachable states (**#reachable states**). In the experiment for the safety specification, we observe that the number of reachable states for LTL model checking is twice as large as for CTL model checking. The increase in allocated OBDD nodes and run time is less than 10%. In the experiments for the liveness specification, the number of the reachable states is four times larger for LTL model checking, while the increase in space and time is 1.5–3 times larger.

The second example is a synchronous bus arbiter which is described in McMillan's thesis [16]. This circuit is composed of a *daisy chain* of identical arbiter cells. The requester with the highest priority receives an acknowledgement from the arbiter under normal operation, while a round-robin scheme is applied when the bus traffic becomes very heavy. Each cell is modeled by a deterministic machine, so the whole arbiter circuit is also a deterministic machine. The specifications in this case are essentially the same as in the case of the DME circuit discussed previously:

1. (*Safety*) No two users are acknowledged simultaneously.

-- Kripke structure

MODULE

$\vdots$

MODULE

$\vdots$

MODULE main

$\vdots$

-- Tableau for f

VAR -- new variables

$\mathtt{EL}\mathbf{X}_{g_1}$: boolean;
$\mathtt{EL}\mathbf{X}_{g_2}$: boolean;
$\vdots$
$\mathtt{EL}\mathbf{X}_{g_N}$: boolean;

DEFINE -- characteristic function

$\mathtt{S}_{h_1}$:= $\cdots$;
$\mathtt{S}_{h_2}$:= $\cdots$;
$\vdots$
$\mathtt{S}_{h_M}$:= $\cdots$;

TRANS -- transition relation

($\mathtt{S}\mathbf{X}_{g_1}$ = next ($\mathtt{S}_{g_1}$)) &
($\mathtt{S}\mathbf{X}_{g_2}$ = next ($\mathtt{S}_{g_2}$)) &
$\vdots$
($\mathtt{S}\mathbf{X}_{g_N}$ = next ($\mathtt{S}_{g_N}$))

-- fairness constraints

FAIRNESS !$\mathtt{S}_{g'_1 \mathbf{U} h'_1}$ | $\mathtt{S}_{h'_1}$
FAIRNESS !$\mathtt{S}_{g'_2 \mathbf{U} h'_2}$ | $\mathtt{S}_{h'_2}$
$\vdots$
FAIRNESS !$\mathtt{S}_{g'_3 \mathbf{U} h'_3}$ | $\mathtt{S}_{h'_3}$

-- new specification

SPEC !($\mathtt{S}_{\neg f}$ & EG true)

Figure 6: Translator output for SMV program

This formula is equivalent to the CTL formula:

$$\mathbf{AG}(\mathrm{req}_1 \rightarrow \mathbf{AF}\,\mathrm{ack}_1)$$

If infinite delays are allowed at each gate, these formulas are not true. In order to overcome this problem we use a fairness constraint which ensures that the output of the gate is reevaluated infinitely often.

SMV provides several options to perform model checking. We verified the circuit using the following approach.

```
TRANS
 ( S_{X g1} = next (S_{g1}) ) &
 ( S_{X g2} = next (S_{g2}) ) &
              ⋮
  ( S_{X gN} = next (S_{gN}) )
```

Likewise, the translator traverses the parse tree and generates an SMV `FAIRNESS` constraint for each node associated with a formula of form $g\,\mathbf{U}\,h$:

```
FAIRNESS  !S_{g U h} | S_h
```

Finally, the translator generates an SMV `SPEC` statement. From Theorem 2, it is clear that the formula $\mathbf{E}\,\neg f$ can be checked using the the specification $S_{\neg f} \wedge \mathbf{EG}\,True$. Thus, in order to check the LTL formula $\mathbf{A}\,f = \neg\,\mathbf{E}\,\neg f$, the translator constructs an SMV `SPEC` statement for $\neg(S_{\neg f} \wedge \mathbf{EG}\,True)$.

We illustrate the translation procedure by applying it to the simple example in Figure 4. The result of this procedure is shown in Figure 7. The statements in lines 1 through 8 come from the original SMV program, while the statements in lines 9 through 19 are generated by the tableau construction for $a\,\mathbf{U}\,b$. The translation procedure first determines that a, b and $\mathbf{X}(a\,\mathbf{U}\,b)$ are elementary formulas and causes the state variable `EL_X_a_U_b` to be declared for $\mathbf{X}(a\,\mathbf{U}\,b)$ in line 10. Next, the DEFINE statement in lines 12 through 16 is constructed for the characteristic functions of $sat(a)$, $sat(b)$, $sat(\mathbf{X}(a\,\mathbf{U}\,b))$, $sat(a\,\mathbf{U}\,b)$ and $sat(\neg a\,\mathbf{U}\,b)$. The Trans statement in line 17 causes the transition relation for the tableau to be constructed, and line 18 contains the fairness constraint for $a\,\mathbf{U}\,b$. Finally, the specification to be checked is given by the 'SPEC' statement in line 19.

7 Experimental Results

7.1 LTL Model Checking and CTL Model Checking

This section describes the experimental results that we obtained for symbolic LTL model checking. In order to compare the performance of LTL model checking with CTL model checking, we used two sequential circuit designs whose specifications can be described in both LTL and CTL.

The first example is a distributed mutual exclusion(DME) circuit designed by Alain Martin[15]. The DME circuit is a speed-independent token ring, which consists of identical arbiter cells. A user of the DME circuit obtains exclusive access to the resource via request and acknowledge signals. We assume aribitrary delay for all gates in the circuit. Each gate is modeled as a finite-state machine that non-deterministically decides either to recompute its output or remain unchanged. We verify the correctness of the following two specifications:

1. (*Safety*) No two users are acknowledged simultaneously.
2. (*Liveness*) All requests are eventually acknowledged.

The safety specification is given by the formula

$$\mathbf{AG} \bigwedge_{1\le i<j\le n} \neg(\mathrm{ack}_i \wedge \mathrm{ack}_j),$$

where ack_i means that user i is acknowledged. This formula is both an LTL formula and a CTL formula. In the experiments for this specification, infinite delays are allowed at each gate. In other words, the output value of each gate can remain unchanged forever.

Next, we verify that requests are eventually acknowledged. We only check this specification with respect to a single user (user 1). In this case the LTL specification has the form:

$$\mathbf{AG}(\mathrm{req}_1 \rightarrow \mathbf{F}\,\mathrm{ack}_1)$$

```
MODULE main  -- simple program

VAR

a: boolean;
b: boolean;

TRANS ( a & !b) -> next(!(a & !b))
TRANS ( a &  b) -> next(a & !b)
TRANS (!a &  b) -> next(!a & b)
TRANS (!a & !b) -> next(!a & b)

SPEC A[a U b]
```

Figure 4: Simple SMV program

```
-- Kripke structure

MODULE
                ⋮
                ⋮
MODULE
                ⋮
                ⋮
MODULE          main
                ⋮
                ⋮
```

```
-- LTL formula
SPEC            A f
```

Figure 5: An SMV program

In the first step, the formula f is negated and expanded to a formula in which the only operators are $\vee$, $\neg$, $\mathbf{X}$, $\mathbf{U}$. The parse tree of $\neg f$ is traversed to find its elementary formulas. If a node associated with formula $\mathbf{X}\,g$ (or $g\,\mathbf{U}\,h$) is visited, then then the corresponding elementary formula $\mathbf{X}\,g$ (or $\mathbf{X}(g\,\mathbf{U}\,h)$) is stored in the list *el_list*. The translator declares a new variable $\mathtt{EL}_{\mathbf{X}\,g}$ for each formula $\mathbf{X}\,g$ in the list *el_list*. Since atomic propositions are already declared in the original SMV program, they are not declared again.

In order to generate descriptions for the transition relation and the fairness constraints, we have to construct the characteristic function S_h of $sat(h)$ for each subformula or elementary formula h in $\neg f$. The translator builds these functions using a `DEFINE` statement[1]. The translator traverses the parse tree of $\neg f$, and generates the appropriate SMV statements at each node.

$\mathtt{S}_h$ `:=` p`;`	if p is an atomic proposition.
$\mathtt{S}_h$ `:=` $\mathtt{EL}_h$`;`	if h is elementary formula $\mathbf{X}\,g$ in *el_list*.
$\mathtt{S}_h$ `:= !`$\mathtt{S}_g$`;`	if $h = \neg g$.
$\mathtt{S}_h$ `:=` $\mathtt{S}_{g_1}$ `|` $\mathtt{S}_{g_2}$`;`	if $h = g_1 \vee g_2$.
$\mathtt{S}_h$ `:=` $\mathtt{S}_{g_2}$ `|` $\big(\mathtt{S}_{g_1}$ `&` $\mathtt{S}_{\mathbf{X}(g_1 \mathbf{U} g_2)}\big)$`;`	if $h = g_1\,\mathbf{U}\,g_2$.

The transition relation can be described in terms of the characteristic functions as follows:

$$\bigwedge_{\mathbf{X}g \in el(f)} S_{\mathbf{X}\,g}(\bar{v}) \Leftrightarrow S_g(\bar{v}')$$

The expression $S_g(\bar{v}')$ is represented in SMV by $\mathtt{next(S}_g\mathtt{)}$. The translator constructs a formula $\mathtt{S}_{\mathbf{X}\,g}$ `=` `next` $\mathtt{(S}_g\mathtt{)}$ for each $\mathbf{X}\,g$ in *el_list*. These formulas are combined in a TRANS statement to give the transition relation for the tableau.

[1] This statement associates a symbol with an SMV expression. When the symbol appears in the program, it is replaced with the expression.

The symbolic model checking algorithm computes the set V as the greatest fixpoint of the following function $\tau : \mathcal{P}(S) \to \mathcal{P}(S)$:

> $\tau(Z) = \{s \mid$ for any c_i, there exists a sequence of length one or greater from s to a state $s' \in sat(c_i) \cap Z\}$.

It is easy to see that, if V is computed in this manner, then every state $\sigma \in V$ will be the beginning of an infinite path that satisfies all of the fairness constraints $c_1, c_2, \ldots, c_l$. Suppose that a state σ is in V. Since $\sigma \in V = \tau(V)$, the definition of τ guarantees that there exists a sequence of length one or larger from σ to some state $\sigma_1 \in sat(c_1)$. Since $\sigma_1 \in V$, we can find a finite sequence from σ_1 to some $\sigma_2 \in sat(c_2)$. Thus, we can eventually find an infinite path π from σ which goes through states in $sat(c_i)$ infinitely often, for each c_i. The states in V are represented by boolean vectors of the form $(\bar{p}, \bar{q}, \bar{r})$. Thus, a state $(\bar{p}, \bar{q})$ in M satisfies $\mathbf{E}\, f$ if and only if there exists $\bar{r}$ such that $(\bar{p}, \bar{q}, \bar{r}) \in V$ and $(\bar{p}, \bar{r}) \in sat(f)$.

6 LTL Model Checking Using the SMV Model Checker

As stated in Section 5, LTL model checking can be reduced to CTL model checking under fairness constraints. If the tableau and the fairness constraints for a given LTL formula are represented implicitly as boolean formulas, we can perform symbolic LTL model checking using an existing symbolic model checker for CTL. We have developed a translator that enables the SMV model checker to handle LTL formulas. For a given LTL formula, the translator generates an SMV program for the corresponding tableau and fairness constraints. We can perform symbolic LTL model checking using the resulting SMV program. In this section, we describe how the translator works.

We begin with a brief description of the SMV model checker. SMV is a tool for checking that finite-state systems satisfy specifications given in CTL. It uses the OBDD-based symbolic model checking algorithm in Section 4. The language component of SMV is used to describe complex finite-state systems. Figure 4 shows an SMV program for the Kripke structure in Figure 2 and an specification $\mathbf{A}(a\ \mathbf{U}\ b)$. This example illustrates the basic features of SMV that are needed to explain the translation procedure. The syntax and semantics of the complete language are given in McMillan's thesis [16].

SMV users can decompose the description of a complex finite-state system into modules. Module definitions begin with the keyword `MODULE`. The module `main` is the top-level module. (The example in Figure 4 contains a single module; however, our translator can handle programs with multiple modules.) Variables are declared using the keyword `VAR`. In the example, `a` and `b` are boolean variables (line 3–4). The `TRANS` statements are used to define transitions of the model (lines 5–8). In the `TRANS` statements, `next(g)` is obtained from g by replacing each state variable v in g by the corresponding next state variable v'. For example, `next(a & !b)` means $a' \wedge \neg b'$ where a' are b' are the next state variables for a and b, respectively. Thus, each `TRANS` statement determines a propositional formula that relates the original state variables and the next state variables. The transition relation for an SMV program is obtained by taking the conjunction of these formulas. CTL formulas are declared as specifications using the keyword `SPEC` (line 9).

Next, we describe the translation algorithm. Suppose that we have an SMV program with an LTL formula $\mathbf{A}\, f$, instead of a CTL formula, as its specification. As stated in Section 5, it is sufficient to handle a formula $\mathbf{E}\, \neg f$. The translator replaces $\mathbf{A}\, f$ with an SMV description of the tableau and the fairness constraints for $\neg f$. The translation of the SMV program in Figure 5 is shown in Figure 6. The translation follows the general procedure outlined in Section 5:

1. Associate a state variable with each elementary formula of $\neg f$.

2. Represent the transition relation of the tableau for $\neg f$ as a boolean formula in terms of the state variables.

3. Represent fairness constraints as boolean formulas in terms of the state variables.

4. Generate a CTL specification.

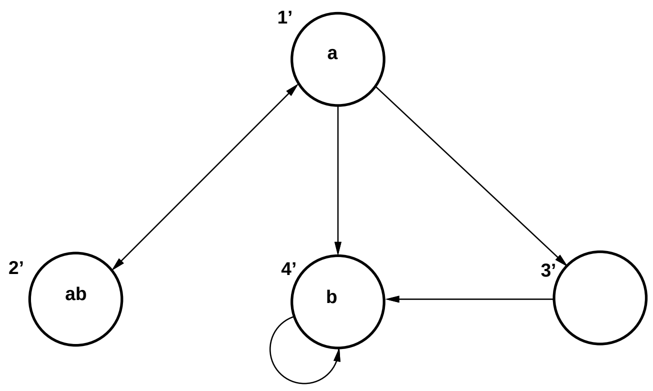

Figure 2: Kripke Structure M

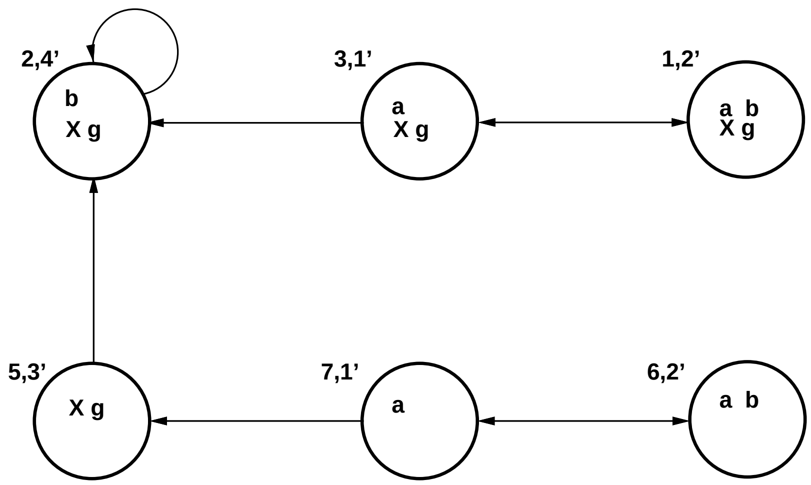

Figure 3: The product P of the structure M and the tableau T

constraint corresponds to the following set of states $\{(2,4'),(5,3'),(7,1'),(6,2'),(1,2')\}$. Thus, every state in Figure 3 satisfies $\mathbf{EG}\,true$. However, only $(2,4')$, $(3,1')$, $(1,2')$ and $(6,2')$ are in $sat(g)$, so the states $1'$, $2'$, and $4'$ of M satisfy $\mathbf{E}\,g = \mathbf{E}[a\,\mathbf{U}\,b]$.

We now describe how the above procedure can be implemented using OBDDs. We assume that the transition relation for M is represented by an OBDD as in the previous section. In order to represent the transition relation for T in terms of OBDDs, we associate with each elementary formula g a state variable v_g. We describe the transition relation R_T as a boolean formula in terms of two copies $\bar{v}$ and $\bar{v}'$ of the state variables. The boolean formula is converted to an OBDD to obtain a concise representation of the tableau. When the composition P is constructed, it is convenient to separate out the state variables that appear in AP_f. The symbol $\bar{p}$ will be used to denote a boolean vector that assigns truth values to these state variables. Thus, each state in S_T will be represented by a pair $(\bar{p},\bar{r})$, where $\bar{r}$ is a boolean vector that assigns values to the state variables that appear in the tableau but not in AP_f. A state in S_M will be denoted by a pair $(\bar{p},\bar{q})$ where $\bar{q}$ is a boolean vector that assigns values to the state variables of M which are not mentioned in f. Thus, the transition relation R_P for the product of the two Kripke structures will be given by

$$R_P(\bar{p},\bar{q},\bar{r},\bar{p}',\bar{q}',\bar{r}') = R_T(\bar{p},\bar{r},\bar{p}',\bar{r}') \wedge R_M(\bar{p},\bar{q},\bar{p}',\bar{q}').$$

We use the symbolic model checking algorithm that handles fairness constraints to find the set of states V that satisfy $\mathbf{EG}\,true$ with the fairness constraints given in (1). We must be careful because of the sequences in the product P that terminates in deadend states. Let $\{c_1, c_2, \ldots, c_l\}$ be fairness constaints.

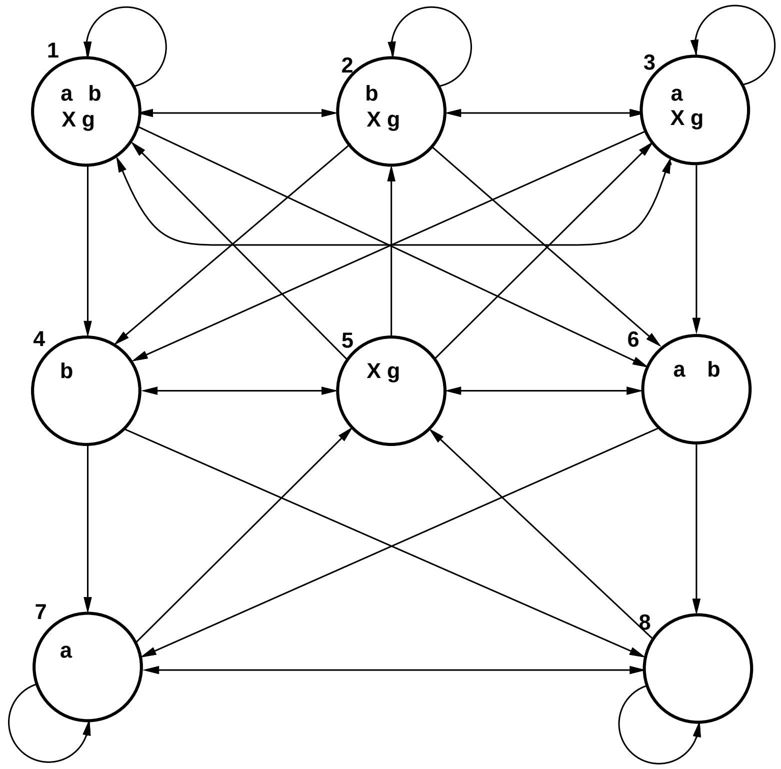

Figure 1: Tableau for $a\,\mathbf{U}\,b$

- $R((\sigma,\sigma'),(\tau,\tau'))$ iff $R_T(\sigma,\tau)$ and $R_M(\sigma',\tau')$.
- $L((\sigma,\sigma')) = L_T(\sigma)$.

P may have deadend states, even if T contains no deadend states. However, it is not difficult to show that P contains exactly the infinite paths π'' for which there are infinite paths π in T and π' in M such that $label(\pi'') = label(\pi) = label(\pi') \mid_{AP_f}$. Thus, if we remove the deadend states from the product, no path that satisfies f will be elminated. As a result, we can safely ignore finite sequences in P. We extend the function sat to be defined over the set of states of the product P by $(\sigma,\sigma') \in sat(g)$ if and only if $\sigma \in sat(g)$.

We next apply CTL model checking and find the set of all states V in P, $V \subseteq sat(f)$, that satisfy $\mathbf{EG}\,true$ with the fairness constraints

$$\{sat(\neg(g\,\mathbf{U}\,h) \vee h) \mid g\,\mathbf{U}\,h \text{ occurs in } f\}. \tag{1}$$

Each of the states in V is in $sat(f)$. Moreover, it is the start of an infinite path that satisfies all of the fairness constraints. These paths have the property that no subformula $g\,\mathbf{U}\,h$ holds almost always on the path while h remains false. The correctness of our construction is summarized by the following theorem.

Theorem 2 *$M,\sigma' \models \mathbf{E}\,f$ if and only if there is a state σ in T such that $(\sigma,\sigma') \in sat(f)$ and $P,(\sigma,\sigma') \models$ $\mathbf{EG}\,True$ under fairness constraints $\{sat(\neg(g\,\mathbf{U}\,h) \vee h) \mid g\,\mathbf{U}\,h$ occurs in $f\}$.*

The proof of this theorem is also given in the Appendix.

To illustrate this construction, we check the formula $g = a\,\mathbf{U}\,b$ on the Kripke structure M in Figure 2. The tableau T for this formula is given in Figure 1. If we compute the product P as described above, we obtain the Kripke structure shown in Figure 3. Although P contains deadend states $(4,4')$ and $(8,3')$, we can ignore those states. We use the CTL model checking algorithm to find the set V of states in $sat(g)$ that satisfy the formula $\mathbf{EG}\,true$ with the fairness constraint $sat(\neg(a\,\mathbf{U}\,b) \vee b)$. It is easy to see that the fairness

Thus, the set of states S_T of the tableau is $\mathcal{P}(el(f))$. The labeling function L_T is defined so that each state is labeled by the set of atomic propositions contained in the state.

In order to construct the transition relation R_T, we need an additional function sat that associates with each elementary subformula g of f a set of states in S_T. Intuitively, $sat(g)$ will be the set of states that satisfy g.

- $sat(g) = \{\sigma \mid g \in \sigma\}$ where $g \in el(f)$.
- $sat(\neg g) = \{\sigma \mid \sigma \notin sat(g)\}$.
- $sat(g \vee h) = sat(g) \cup sat(h)$.
- $sat(g\, \mathbf{U}\, h) = sat(h) \cup \big(sat(g) \cap sat(\mathbf{X}(g\, \mathbf{U}\, h))\big)$.

We want the transition relation to have the property that each elementary formula in a state is true in that state. Clearly, if $\mathbf{X}g$ is in some state σ, then all the successors of σ should satisfy g. Furthermore, since we are dealing with LTL formulas, if $\mathbf{X}g$ is not in σ, then σ should satisfy $\neg\mathbf{X}g$. Hence, no successor of σ should satisfy g. The obvious definition for R_T is

$$R_T(\sigma, \sigma') = \bigwedge_{\mathbf{X}g \in el(f)} \sigma \in sat(\mathbf{X}\, g) \Leftrightarrow \sigma' \in sat(g).$$

Figure 1 gives the tableau for the formula $g = a\, \mathbf{U}\, b$. To reduce the number of edges, we connect two states σ and σ' with a bidirectional arrow if there is an edge from σ to σ' and also from σ' to σ. Each subset of $el(g)$ is a state of T. $sat(\mathbf{X}g) = \{1, 2, 3, 5\}$ since each of these states contains the formula $\mathbf{X}g$. $sat(g) = \{1, 2, 3, 4, 6\}$ since each of these states either contains b or contains a and $\mathbf{X}g$. There is a transition from each state in $sat(\mathbf{X}g)$ to each state in $sat(g)$ and from each state in the complement of $sat(\mathbf{X}g)$ to each state in the complement of $sat(g)$.

Unfortunately, the definition of R_T does not guarantee that *eventuality* properties are fulfilled. We can see this behavior in Figure 1. Although state 3 belongs to $sat(g)$, the path that loops forever in state 3 does not satisfy the formula g since b never holds on that path. Consequently, an additional condition is necessary in order to identify those paths along which f holds. A path π that starts from a state $\sigma \in sat(f)$ will satisfy f if and only if

- For every subformula $g\, \mathbf{U}\, h$ of f and for every state σ on π, if $\sigma \in sat(g\, \mathbf{U}\, h)$ then either $\sigma \in sat(h)$ or there is a later state τ on π such that $\tau \in sat(h)$.

The definition of R_T can also cause T to have states which have no successors. We call such states *deadend* states. For example, a state $\{a, \mathbf{X}\, a, \mathbf{X}\, \neg a\}$ has no successors, because no state is in both $sat(a)$ and $sat(\neg a)$. Because of the semantics of LTL, all of the sequences that satisfy a path formula f must be infinite. Thus, if we remove the deadend states from the tableau of f, no path that satisfies f will be elminated. Therefore, in the tableau of f, we can safely ignore finite sequences that terminate in deadend states.

In order to state the key property of the tableau construction, we must introduce some new notation. Let $\pi = s_0, s_1, \ldots$ be a path in a Kripke structure M, then $label(\pi) = L(s_0), L(s_1), \ldots$. Let $l = l_0, l_1, \ldots$ be a sequence of subsets of some set Σ and let $\Sigma' \subseteq \Sigma$. The *restriction* of l to Σ', denoted by $l\,|_{\Sigma'}$, is the sequence $l'_0, l'_1, \ldots$ where $l'_i = l_i \cap \Sigma'$ for every $i \geq 0$. The following theorem makes precise the intuitive claim that T includes every path which satisfies f.

Theorem 1 *Let T be the tableau for the path formula f. Then, for every Kripke structure M and every path π' of M, if $M, \pi' \models f$ then there is a path π in T that starts in a state in $sat(f)$, such that $label(\pi')\,|_{AP_f} = label(\pi)$.*

We prove this theorem in the Appendix.

Next, we want to compute the product $P = (S, R, L)$ of the tableau $T = (S_T, R_T, L_T)$ and the Kripke structure $M = (S_M, R_M, L_M)$.

- $S = \{(\sigma, \sigma') \mid \sigma \in S_T, \sigma' \in S_M \text{ and } L_M(\sigma') \cap AP_f = L_T(\sigma)\}$.

4 CTL Model Checking

CTL Model checking is the problem of finding the set of states in a state transition graph where a given CTL formula is true. One approach for solving this problem is a symbolic model checking using an OBDD to represent the transition relation of the graph. Assume that the transition relation is given as a boolean formula $R(\bar{v}, \bar{v}')$ in terms of current state variables $\bar{v} = (v_1, \ldots, v_n)$ and next state variables $\bar{v}' = (v'_1, \ldots, v'_n)$. The algorithm takes a CTL formula f, and the OBDD that represents $R(\bar{v}, \bar{v}')$. For each subformula g, the algorithm computes the states that satisfy g in a bottom-up manner. This step is performed by OBDD operations. The algorithm returns an OBDD that represents exactly those states of the system that satisfy the formula f.

Fairness constraints were introduced for checking the correctness of CTL formulas along fair computation paths. A *fairness constraint* can be an arbitrary set of states, usually described by a formula of the logic. A path is said to be *fair* with respect to a set of fairness constraints if each constraint holds *infinitely often* along the path. The path quantifiers in CTL formulas are then restricted to fair paths. The CTL model checking under given fairness constraints can also be performed using OBDD operations. As will be shown in the next section, LTL model checking can be reduced to CTL model checking under fairness constraints.

5 LTL Model Checking

In this section we consider the model checking problem for linear temporal logic. Let $\mathbf{A}\, f$ be a linear temporal logic formula. Thus, f is a *restricted path formula* in which the only state subformulas are atomic propositions. We wish to determine all of those states $s \in S$ such that $M, s \models \mathbf{A}\, f$. By definition $M, s \models \mathbf{A}\, f$ iff $M, s \models \neg\, \mathbf{E} \neg f$. Consequently, it is sufficient to be able to check the truth of formulas of the form $\mathbf{E}\, f$ where f is a restricted path formula. If the Kripke structure is represented explicitly as a state transition graph, this problem is known to be PSPACE-complete [17] in general.

Lichtenstein and Pnueli [14] developed an algorithm for the problem that was linear in the size of the model M and exponential in the length of the formula f. Although their algorithm was linear in the size of the model, it was still impractical for large examples because of the state explosion problem. As in the case of CTL model checking, representing the transition relation as an OBDD enables the procedure to be applied to much larger examples. The exponential complexity of their algorithm in terms of formula length is caused by a tableau construction which may require exponential space in the size of the formula.

Burch et. al developed a symbolic satisfiability algorithm for LTL [4], This algorithm is based on implicit tableau construction, which leads to an additional reduction in space and time. We also use this implicit technique in the following model checking algorithm. We begin with an informal description of the algorithm. Given a formula $\mathbf{E}\, f$ and a Kripke structure M, we construct a special Kripke structure T called the *tableau* for the path formula f. This structure includes *every* path that satisfies f. By composing T with M, we find the set of paths that appear in both T and M. A state in M will satisfy $\mathbf{E}\, f$ if and only if it is the start of a path in the composition that satisfies f. The CTL model checking procedure described in Section 4 is used to find these states.

We now describe the construction of the tableau T in detail. Let AP_f be the set of atomic propositions in f. The tableau associated with f is a structure $T = (S_T, R_T, L_T)$ with AP_f as its set of atomic propositions. Each state in the tableau is a set of *elementary* formulas obtained from f. The set of elementary subformulas of f is denoted by $el(f)$ and is defined recursively as follows:

- $el(p) = \{p\}$ if $p \in AP$.
- $el(\neg g) = el(g)$.
- $el(g \vee h) = el(g) \cup el(h)$.
- $el(\mathbf{X}\, g) = \{\mathbf{X}\, g\} \cup el(g)$.
- $el(g\, \mathbf{U}\, h) = \{\mathbf{X}(g\, \mathbf{U}\, h)\} \cup el(g) \cup el(h)$.

- If f and g are state formulas, then $\neg f$ and $f \vee g$ are state formulas.
- If f is a path formula, then $\mathbf{E}(f)$ is a state formula.

Two additional rules are needed to specify the syntax of path formulas:

- If f is a state formula, then f is also a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $\mathbf{X}\, f$, and $f\, \mathbf{U}\, g$ are path formulas.

CTL* is the set of state formulas generated by the above rules.

We define the semantics of CTL* with respect to a Kripke structure $M = \langle S, R, L \rangle$, where S is the set of states; $R \subseteq S \times S$ is the transition relation, which must be *total* (i.e., for all states $s \in S$ there exists a state $s' \in S$ such that $(s, s') \in R$); and $L : S \rightarrow \mathcal{P}(AP)$ is a function that labels each state with a set of atomic propositions true in that state. In this paper, we assume that all Kripke structures are *finite*.

A *path in* M is an infinite sequence of states, $\pi = s_0, s_1, \ldots$ such that for every $i \geq 0$, $(s_i, s_{i+1}) \in R$. We use π^i to denote the *suffix* of π starting at s_i. If f is a state formula, the notation $M, s \models f$ means that f holds at state s in the Kripke structure M. Similarly, if f is a path formula, $M, \pi \models f$ means that f holds along path π in Kripke structure M. When the Kripke structure M is clear from context, we will usually omit it. The relation $\models$ is defined inductively as follows (assuming that f_1 and f_2 are state formulas and g_1 and g_2 are path formulas):

1. $s \models p \quad \Leftrightarrow p \in L(s)$.
2. $s \models \neg f_1 \quad \Leftrightarrow s \not\models f_1$.
3. $s \models f_1 \vee f_2 \quad \Leftrightarrow s \models f_1$ or $s \models f_2$.
4. $s \models \mathbf{E}(g_1) \quad \Leftrightarrow$ there exists a path π starting with s such that $\pi \models g_1$.
5. $\pi \models f_1 \quad \Leftrightarrow s$ is the first state of π and $s \models f_1$.
6. $\pi \models \neg g_1 \quad \Leftrightarrow \pi \not\models g_1$.
7. $\pi \models g_1 \vee g_2 \quad \Leftrightarrow \pi \models g_1$ or $\pi \models g_2$.
8. $\pi \models \mathbf{X}\, g_1 \quad \Leftrightarrow \pi^1 \models g_1$.
9. $\pi \models g_1\, \mathbf{U}\, g_2 \quad \Leftrightarrow$ there exists a $k \geq 0$ such that $\pi^k \models g_2$ and for all $0 \leq j < k$, $\pi^j \models g_1$.

The following abbreviations are used in writing CTL* formulas:

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$
- $\mathbf{A}(f) \equiv \neg\, \mathbf{E}(\neg f)$
- $\mathbf{F}\, f \equiv \mathit{true}\, \mathbf{U}\, f$
- $\mathbf{G}\, f \equiv \neg\, \mathbf{F} \neg f$

CTL [1, 7] is a restricted subset of CTL* that permits only branching-time operators—each of the linear-time operators $\mathbf{G}$, $\mathbf{F}$, $\mathbf{X}$, and $\mathbf{U}$ must be immediately preceded by a path quantifier. More precisely, CTL is the subset of CTL* that is obtained if the following two rules are used to specify the syntax of path formulas.

- If f and g are state formulas, then $\mathbf{X}\, f$ and $f\, \mathbf{U}\, g$ are path formulas.
- If f is a path formula, then so is $\neg f$.

Linear temporal logic (LTL), on the other hand, will consist of formulas that have the form $\mathbf{A}\, f$ where f is a path formula in which the only state subformulas permitted are atomic propositions. More precisely, a path formula is either:

- an atomic proposition $p \in AP$.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $\mathbf{X}\, f$, and $f\, \mathbf{U}\, g$ are path formulas.

There are eight basic CTL operators: **AX**, **EX**, **AG**, **EG**, **AF**, **EF**, **AU** and **EU**. Each of the eight operators can be expressed in terms of three operators **EX**, **EG**, and **EU**.

checking an appropriate CTL formula on P we can find the set V_f of all of those states s such that f holds along every path that begins at s. The projection of V_f to the state variables of M gives the set of states where the formula f holds.

Note that our approach makes it unnecessary to modify SMV (or even understand how SMV is actually implemented). We have evaluated the approach on several standard SMV programs (including Martin's distributed mutual exclusion circuit [15] and the synchronous arbiter described in McMillan's thesis [16]). In order to compare the performance of LTL model checking with CTL model checking, we chose specifications which could be expressed in both CTL and LTL. For the examples we considered, the LTL model checker required at most twice as much time and space as the CTL model checker. We also verified non-trivial LTL formulas to examine the feasibility of LTL model checking. The amount of time and space that is required is quite reasonable. Based on the results that we have obtained, it appears that efficient LTL model checking is possible when the specifications are not excessively complicated.

2 Binary Decision Diagrams

Ordered binary decision diagrams (OBDDs) are a canonical form representation for boolean formulas [3]. They are often substantially more compact than traditional normal forms such as conjunctive normal form or disjunctive normal form, and they can be manipulated very efficiently. An OBDD is similar to a binary decision tree, but has the following properties.

- Its structure is a directed acyclic graph rather than a tree.
- A total order is placed on the occurrence of variables as the graph is traversed from root to leaf.
- No two subgraphs in the graph represents the same function.

Bryant showed that given a variable ordering, the OBDD representation for a boolean formula is unique.

We can implement various important logical operations using OBDDs. The function that restricts some argument x_i of the boolean function f to a constant value b, denoted by $f\mid_{x_i \leftarrow b}$, can be performed in time which is linear in the size of the original binary decision diagram [3]. The restriction algorithm allows us to compute the OBDD for the formula $\exists x f$ as $f\mid_{x\leftarrow 0} + f\mid_{x\leftarrow 1}$. All 16 two-argument logical operations can also be implemented efficiently on boolean functions that are represented as OBDDs. The complexity of these operations is linear in the size of the argument OBDDs [3]. Furthermore equivalence checking of two boolean functions can be done in constant time, by using a hash table properly[2].

OBDDs are extremely useful for obtaining concise representations of relations over finite domains [4, 16]. If R is n-ary relation over $\{0, 1\}$ then R can be represented by the OBDD for its *characteristic function*

$$f_R(x_1, \ldots, x_n) = 1 \text{ iff } R(x_1, \ldots, x_n).$$

Otherwise, let R be an n-ary relation over the finite domain D. Using an appropriate binary encoding of D, we can represent R by an OBDD.

3 Computation Tree Logics

We begin by describing the temporal logic CTL* [7, 8, 12], which can express both linear-time and branching-time properties. In this logic, a path quantifier, either **A** ("for all computation paths") or **E** ("for some computation paths") can prefix an assertion composed of arbitrary combinations of the usual linear-time operators **G** ("always"), **F** ("sometimes"), **X** ("nexttime"), and **U** ("until"). Both Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) are included in CTL* .

There are two types of formulas in CTL* : *state formulas* (which are true in a specific state) and *path formulas* (which are true along a specific path). Let AP be the set of atomic proposition names. The syntax of state formulas is given by the following rules:

- If $p \in AP$, then p is a state formula.

Another Look at LTL Model Checking *

E. Clarke[1], O. Grumberg[2], K. Hamaguchi[3]

1. Carnegie Mellon University, Pittsburgh

2. The Technion, Haifa

3. Kyoto University, Kyoto

Abstract

We show how LTL model checking can be reduced to CTL model checking with fairness constraints. Using this reduction, we also describe how to construct a *symbolic* LTL model checker that appears to be quite efficient in practice. In particular, we show how the SMV model checking system developed by McMillan [16] can be extended to permit LTL specifications. The results that we have obtained are quite surprising. For the specifications which can be expressed in both CTL and LTL, the LTL model checker required at most twice as much time and space as the CTL model checker. We also succeeded in verifying non-trivial LTL specifications. The amount of time and space that is required is quite reasonable. Based on the examples that we considered, it appears that efficient LTL model checking is possible when the specifications are not excessively complicated.

Keywords: automatic verification, temporal logic, model checking, binary decision diagrams

1 Introduction

Over the past thirteen years there has been considerable research on efficient model checking algorithms for branching-time temporal logics like CTL (See [10] for a survey). Verification tools based on these algorithms have discovered non-trivial design errors in sequential circuits and protocols [9] and are now beginning to be used in industry. There has been relatively little research, however, on efficient model checking algorithms for linear-temporal logic (LTL), and practical verification tools are virtually non-existent. In fact, the question of whether it is possible to develop such tools has been argued for many years. Sistla and Clarke [17] showed in 1982 that the model checking problem for LTL was, in general, PSPACE complete. Later, Pnueli and Lichtenstein [14] gave an LTL model checking algorithm that was exponential in the size of the formula, but *linear* in the size of the model. Based on this result, they argued that the high complexity of LTL model checking might still be acceptable for short formulas. Vardi and Wolper [18] obtained a different algorithm based on ω-automata with roughly the same complexity. Unfortunately, the LTL algorithms appeared significantly more difficult to implement. Because of this, very few LTL model checkers were actually constructed. To the best of our knowledge, no experiments were made to determine how the CTL and LTL model checking algorithms actually compared in practice.

In this paper we show how LTL model checking can be reduced to CTL model checking with fairness constraints. We also describe how to construct a *symbolic* LTL model checker that appears to be quite efficient in practice. In particular, we show how the SMV model checking system developed by McMillan as part of his Ph.D. thesis [16] can be extended to permit LTL specifications. We have developed a translator $\mathcal{T}$ that takes an LTL formula f and constructs an SMV program $\mathcal{T}(f)$ to build the tableau for f. The tableau construction that we use is similar to the one described in [4]. To check that f holds for some SMV program M, we combine the text of $T = \mathcal{T}(\neg f)$ with the text of M to obtain a new SMV program $P = Prod(T, M)$. We add CTL fairness constraints to P in order to make sure that eventualities of the form $a\,\mathbf{U}\,b$ are actually fulfilled (i.e. to eliminate those paths along which $a\,\mathbf{U}\,b$ and a hold continuously, but b never holds). By

*This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597 and in part by the National Science foundation under Grant No. CCR-9217549 and in part by the Semiconductor Research Corporation under Contract 92-DJ-294 and in part by the Wright Laboratory, Aeronautical Systems Center Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330, and by grant no. 89-413 from the United State-Israel Binational Science Foundation (BSF), Jerusalem, Israel. The third author was supported by a Kurata Research Grant and a Kyoto University Foundation Grant. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the U.S. government.