Improving Temporal Logic Tableaux using Integer Constraints

Reiner Hähnle and Ortrun Ibens

University of Karlsruhe
Institute for Logic, Complexity and Deduction Systems
76128 Karlsruhe, Germany
{reiner,ortrun}@ira.uka.de

1 Introduction

In this position paper we present some ideas that aim to improve analytic tableau for temporal logics with the ultimate goal of reviving the interest in using them for temporal logic satisfiability checking. Although tableau formulations for several propositional temporal logics exist [9], these are not used much in practice, because the tableau size becomes intractable already for quite small formulas. Moreover, checking tableau closure is complicated and expensive to implement. For practical purposes, usually automata theoretic approaches are preferred [3]. It might, however, still be interesting to have competitive tableau formulations as will be pointed out in Section 4.

The ideas for the research reported here were stimulated by [6, 7] where an extension of analytic tableau with linear constraints led to an efficient (mixed) integer programming formulation of a tableau system for finitely and infinitely-valued propositional logics. The usefulness of linear constraints for many-valued logics comes from the fact that the set of truth values can be identified with (a subset of) the integers, hence many-valued connectives can be characterized using integer constraints. If we consider, say, a discrete linear temporal logic without past time operators, then the *always*, *sometimes*, etc. operators realize certain properties of the natural numbers when Kripke worlds are identified with numbers and the linear order on the worlds with the successor function. It is tempting, in our opinion, to try using linear constraints over world variables in order to give more concise tableau systems for such logics than it is possible without the use of constraints. In this note we give one possible formulation of a constraint tableau system for one of the simplest propositional temporal logics, namely for discrete linear temporal logic with (future) always and sometimes.

2 Basic Definitions

Let $\mathcal{L} = \langle L_0, \neg, \vee, \wedge, \Box, \Diamond \rangle$ be a propositional language of *linear temporal logic* (LTL) with negation, disjunction, conjunction, always, and sometimes defined as usual over a non-empty set of propositional variables L_0.

Call $\mathcal{S} = \{k + x_1 + \cdots + x_n | k, n \in \mathbb{N}_0, x_1, \ldots, x_n \in S_0\}$ the set of *signs* over a non-empty set S_0 of world variables disjoint with L_0. The language of *signed linear temporal logic* (S-LTL) is $\mathcal{L}^* = \{s : \phi | s \in \mathcal{S}, \phi \in \mathcal{L}\}$.

An *S–LTL–Interpretation* v maps each propositional variable $p \in L_0$ to the set of time points p holds at and each world variable $s_0 \in S_0$ to a time point. The extension to formulas $f \in \mathcal{L}$ and to signs $s \in \mathcal{S}$ is immediate.

Let $s : \phi \in \mathcal{L}^*$, then $s : \phi$ is *satisfiable* if there is an S-LTL interpretation v such that $v(s) \in v(\phi)$. The extension to sets of S-LTL-formulas is immediate.

$\mathcal{C} = \{s_1 \leq_{\mathbb{N}_0} s_2 | s_1, s_2 \in \mathcal{S}\}$ is the set of S-LTL *constraints*. Usually we omit the subscript of $\leq$. We make use of the standard abbreviations $<, =, \neq$ etc.

3 Constraint Tableau Calculus

We assume the reader is familiar with tableau calculi for propositional logics, and in particular with uniform notation for propositional modal formulas as defined in [5]. Just recall that type α formulas are conjunctive, type β are disjunctive, type ν are universal or 'always', and type π are existential or 'sometimes'.

In contrast to classical propositional tableaux, nodes consist of *sets of signed formulas* rather than single formulas. Moreover, we must accomodate linear constraints. Hence, a *tableau node* K is a pair $\langle \Phi, C \rangle \in (2^{\mathcal{L}^*} \times 2^{\mathcal{C}})$. A node $K = \langle \Phi, C \rangle$ is satisfiable if Φ has a model v which as well solves C.

In Table 1 we give correspondences between premises and conclusions of tableau rules for all types of compound formulas. The superscripts '$\ldots^{\nu}$' in the ν rules are to be understood as markers indicating that no logical rule is to be applied anymore to the formula thus superscribed. This is a crucial difference to conventional temporal tableau systems, and we will see how completeness is restored by different means. There are four types of tableau rules which will be explained in the following.

Logical tableau rules can be specified as in Table 2. Note that taking the disjoint union in the premise implies that the main formula of the premise does not occur anymore in the conclusion. The letter k in the π rules is a new variable from S_0 that occurs not on the current branch. Informally, the conclusion of a π rule says that there is some point in the future when ϕ comes true.

Table 1. Correspondence between premises and conclusions of temporal constraint tableau rules.

α	α_1	α_2
$s : \phi \wedge \psi$	$s : \phi$	$s : \psi$
$s : \neg(\phi \vee \psi)$	$s : \neg\phi$	$s : \neg\psi$
$s : \neg\neg\phi$	$s : \phi$	$s : \phi$

β	β_1	β_2
$s : \phi \vee \psi$	$s : \phi$	$s : \psi$
$s : \neg(\phi \wedge \psi)$	$s : \neg\phi$	$s : \neg\psi$

ν	ν_1	ν_2
$s : \Box\phi$	$s : \phi$	$(s : \Box\phi)^{\nu}$
$s : \neg\Diamond\phi$	$s : \neg\phi$	$(s : \neg\Diamond\phi)^{\nu}$

π	$\pi_1(k)$	$c_1(k)$
$s : \Diamond\phi$	$s + k : \phi$	$k \geq 0$
$s : \neg\Box\phi$	$s + k : \neg\phi$	$k \geq 0$

The rôle of the linear constraints is to record the information on possible models for each branch in a more concise form than in conventional tableaux.

Table 2. Temporal tableau rules for compound formulas.

$$\frac{\langle \{\alpha\} \dot\cup \Phi, C\rangle}{\langle \{\alpha_1, \alpha_2\} \cup \Phi, C\rangle} \qquad \frac{\langle \{\beta\} \dot\cup \Phi, C\rangle}{\langle \{\beta_1\} \cup \Phi, C\rangle \,|\, \langle \{\beta_2\} \cup \Phi, C\rangle}$$

$$\frac{\langle \{\nu\} \dot\cup \Phi, C\rangle}{\langle \{\nu_1, \nu_2\} \cup \Phi, C\rangle} \qquad \frac{\langle \{\pi\} \dot\cup \Phi, C\rangle}{\langle \{\pi_1(k)\} \cup \Phi, \{c_1(k) \cup C\}\rangle}$$

The justification is that every class of counter models can in fact be characterized by a set of linear inequations with integral solutions. One link between formulas and constraints is constituted by the π rules as we have seen, another link is obtained from the observation that, obviously, a formula and its complement cannot be true simultaneously at the same time point. This concept is formalized in the following *conflict rule*.

$$\frac{\langle \{s : \phi, t : \neg\phi\} \cup \Phi, C\rangle}{\langle \{s : \phi, t : \neg\phi\} \cup \Phi, C \cup \{s \neq t\}\rangle}$$

As a matter of fact, there are other possible constraints one could generate, for instance, $s < t$ from $s : \phi$ and $t : \Box\neg\phi$ etc., but these are being subsumed by the *synchronization rule* below which we need for completeness anyway.

In conventional temporal tableau systems such as in [9] two more ingredients are needed for which we have no substitute so far. First, for completeness one must have the information that an always-formula does hold in all subsequent states. Usually this is achieved by reintroducing always-formulas in the conclusion of a rule and thus allowing them to be applied more than once. As a consequence, in order to obtain an effective tableau system, nodes N all whose formulas have occurred already in a previous node M are 'looped back' to this node M such that temporal tableaux really are directed graphs with a unique root. We intend to combine both rules, the 'restart' rule on always-formulas and the 'loop back' rule, into a single rule called *synchronization rule*. The rationale behind it is to 'synchronize' always-formulas so that they are available in the 'present' state. This has a similar effect as the restart rule, but because in the constraint part of a node we can accumulate information from different time points, a finite number of applications of this rule is sufficient.

In slight abuse of notation let now ν denote the unsigned part of an always-formula, and ν_1 the unsigned part of its correspondent according to Table 1; let $\lambda \in L_0 \cup \{\neg p | p \in L_0\}$, i.e. λ is a literal. Then we can write down the synchronization rule as follows:

$$\frac{\langle \{s : \lambda, t : \nu\} \cup \Phi, C\rangle}{\langle \{s : \lambda, t : \nu\} \cup \Phi, C \cup \{s \leq t\}\rangle \;\; | \;\; \langle \{s : \lambda, t : \nu, (s : \nu_1)^{\mathrm{syn}}\} \cup \Phi, C \cup \{s > t\}\rangle}$$

The *syn* marker prevents indefinite application of the synchronization rule, because the synchronization rule can only be applied to formulas without a *syn* marker.

But there are examples where the synchronization rule must be applied to a formula which itself is a synchronization result (see Fig. 1). So we need a rule which activates formulas with a *syn* marker, called *activation rule*. The idea is to activate a literal $(s : \lambda)^{\mathrm{syn}} \in \Phi$ if a literal $s_0 : \lambda \in \Phi$ together with an S–LTL–interpretation v which maps $v(s) = v(s_0)$ cannot be found, and to activate an always formula $(s : \nu)^{\mathrm{syn}} \in \Phi$ if an always formula $s_0 : \nu \in \Phi$ together with an S–LTL–interpretation v which maps $v(s) \geq v(s_0)$ cannot be found. There is an effective test for the applicability of the activation rule, but the details must be omitted due to lack of space. Let $(s : \phi)^{\mathrm{syn}}$ be a formula to be activated. Then the activation rule is simply the following:

$$\frac{\langle \{(s : \phi)^{\mathrm{syn}}\} \cup \Phi, C\rangle}{\langle \{s : \phi\} \cup \Phi, C\rangle}$$

A node $K = \langle \Phi, C\rangle$ in our tableau is *contradictory* if C is unsolvable. A tableau is *closed* if every branch contains a contradictory node. A formula $\phi \in \mathcal{L}$ is valid iff the tableau with root $\langle \{s : \neg\phi\}, \{s \geq 0\}\rangle$, $s \in S_0$, is closed. In Fig. 1 we give a tableau proof of $\neg\Diamond\Box p \vee \neg\Box\Diamond\neg p$.

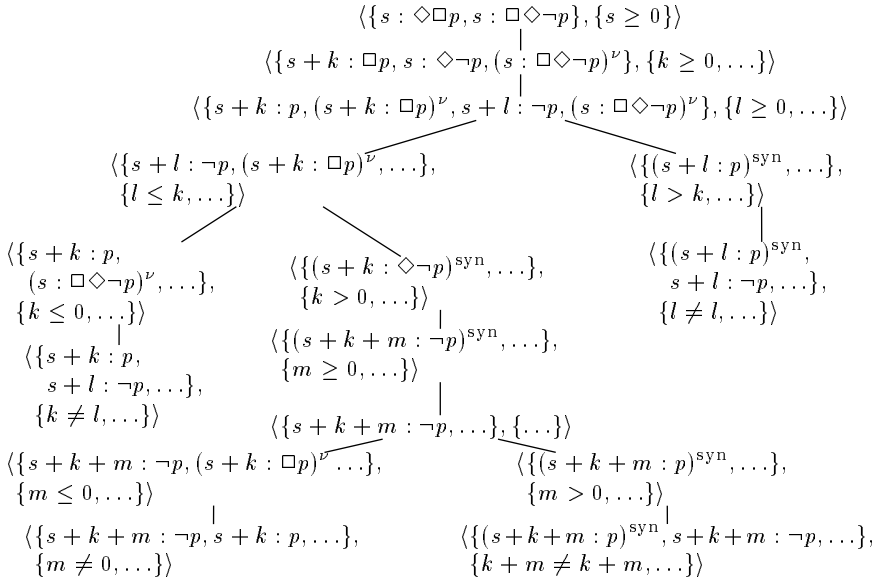

Fig. 1. We omitted the first steps which are purely propositional. The next two steps are straightforward ν and π rule applications. Next follow two applications of the synchronisation rule that split the proof into three branches. The first and the third of these can be closed after a conflict rule application. The second branch can be closed after the activation rule and again the synchronization rule are applied. (Due to space restrictions only such formulas and constraints are displayed that are newly generated or that are needed in that step. The formulas between which a synchronization takes place appear as the first two formulas in the left leaf which is generated by the synchronization rule.)

4 Conclusion

It is fairly easy, if somewhat tedious, to prove that our rules preserve satisfiability. Completeness, however, is a much tougher question, and the proof is quite involved. This is one of the reasons why the present note is merely a position paper. The other reason is that so far we did not show that the idea of incorporating linear constraints into temporal logic tableaux is in some sense a real improvement over the current state of the art. On the other hand, we would like to point out at least a few advantages that might be gained from it:

- Better performance if compared to conventional temporal tableaux. Admittedly, this is not obvious from the present appearance of the rules, but there is ample room for optimisation, for example, in the synchronisation rules.
- Implementing real-time logics like the ones suggested in [1].
- Extend the use of linear constraints to a translation of temporal logic satisfiability into integer programming (as it has been done in [6] for many-valued logic). This would allow easy amalgamation with other non-standard features like multiple truth values (needed, for instance, for hardware verification at the switch level [8]) or non-monotonicity [2] in a homogenous framework.
- Perspective of a constraint-based approach to non-classical deduction if integrated with ideas from [4].
- Lifting to restricted first-order versions.

References

1. R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proc. 5th LICS*, pages 390–401. IEEE Press, 1990.
2. C. Bell, A. Nerode, R. Ng, and V. Subrahmanian. Mixed integer programming methods for computing nonmonotonic deductive databases. *JACM, to appear*, 1994.
3. J. Burch, E. Clarke, K. McMillan, and D. Dill. Sequential circuit verification using symbolic model checking. In *Proc. 27th Design Automation Conference (DAC 90)*, pages 46–51, 1990.
4. M. D'Agostino and D. M. Gabbay. Labelled refutation sustems: A case study. In *Proc. Second Workshop on Theorem Proving with Tableaux and Related Methods, Marseille*. Technical Report, MPI Saarbrücken, 1993.
5. M. C. Fitting. *Proof Methods for Modal and Intutionistic Logics*. Reidel, Dordrecht, 1983.
6. R. Hähnle. A new translation from deduction into integer programming. In J. Calmet and J. A. Campbell, editors, *Proc. Int. Conf. on Artificial Intelligence and Symbolic Mathematical Computing AISMC-1, Karlsruhe, Germany*, pages 262–275. Springer, LNCS 737, 1992.
7. R. Hähnle. *Automated Theorem Proving in Multiple-Valued Logics*, Oxford University Press, 1993.
8. R. Hähnle and W. Kernig. Verification of switch level designs with many-valued logic. In A. Voronkov, editor, *Proc. LPAR'93, St. Petersburg*, pages 158–169. Springer, LNAI 698, 1993.
9. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.