Labelled Tableaux for Temporal Logic with Cardinality Constraints

Clare Dixon, Boris Konev
Department of Computer Science,
University of Liverpool,
Liverpool, UK

Renate A. Schmidt, Dmitry Tishkovsky
School of Computer Science,
University of Manchester,
Manchester, UK

***Abstract*—Frequently when formalising systems that change over time, we must represent statements, coming from physical constraints or representational issues, stating that exactly n literals (or less than n literals) of a set hold. While we can write temporal formulae to represent this information, such formulae both complicate and increase the size of the specification and adversely affect the performance of provers. In this paper, we consider reasoning about problems specified in propositional linear time temporal logics in the presence of such constraints on literals. We present a sound, complete and terminating tableau calculus which embeds constraints into its construction avoiding their explicit evaluation. We use METTEL^2, an automated tableau prover generator, to provide an implementation of the calculus and give experimental results using the prover.**

I. INTRODUCTION

Temporal logics have been used to represent and reason about systems that change over time [4], [12]. Often when representing such systems we need to formalise that exactly n or less than or equal to m propositions from a set hold or do not hold. This may come about for two reasons. First, they may represent real constraints in the world, for example, n machines are available to perform a task or a room has capacity of at most m. Alternatively, they may come about from representational issues. For example, consider a number of robots moving about a grid. Each robot may occupy exactly one square in the grid, so, if $at(i)_{x,y}$ denotes that the robot i is in square (x, y), then exactly one of the propositions $at(i)_{0,0}, at(i)_{0,1}, \ldots at(i)_{n,n}$ holds, for each robot, for an $n \times n$ grid.

One way to deal with such constraints is to rewrite them as temporal formulae, for example, if we are required to make exactly one from the set $\{p, q, r\}$ true we can represent this as the formulae $\square(p \vee q \vee r)$, $\square(\neg p \vee \neg q)$, $\square(\neg p \vee \neg r)$, $\square(\neg q \vee \neg r)$, where $\square$ is the operator 'at every moment in time' from propositional linear-time temporal logic (PTL). However, introducing such additional formulae lengthen and complicate the specification and adversely affect the performance of provers. Instead, we consider a logic, called TLC, introduced in [8]. TLC is a propositional linear-time temporal logic that allows sets of constraints as input. As well as presenting some case studies, [8] provided an algorithm explicitly constructing the Kripke frame representing a temporal model of the given formula. Notice that the satisfiability problem for PTL with just one proposition is already PSPACE-complete [7]; therefore, the complexity of the satisfiability problem for TLC is also PSPACE-complete, in general. However, if we restrict our consideration to formulae in a certain normal form, the size of the associated Kripke frame is exponential in the number of unconstrained propositions and only polynomial in the number of constrained propositions [8]. An implementation of this algorithm showed that using this approach was advantageous for formulae in the normal form, particularly when a large proportion of the propositions were constrained. These experimental results suggested that dealing with constraints expressed explicitly may also be beneficial for TLC formulae not in the normal form. Such formulae were poorly handled by the system described in [8] as the translation into the normal form introduces additional unconstrained propositions. We also identified a number of disadvantages: the need to explicitly enumerate all the sets of propositions satisfying the constraints and that the expensive breadth-first style of construction is undesirable.

The aim of this paper is twofold. First, it builds on and extends the work in [8] and its related implementation in a number of ways. We develop a tableau calculus that includes rules for reasoning about constraints in a more goal-directed, incremental way. This retains the advantages of using constraints and overcomes the disadvantages encountered in the approach from [8]. In particular, we are not always forced to explicitly enumerate the constraints. Further we can construct the tableau derivation branch by branch in a depth-first left-to-right manner.

Second, we are interested in exploring the possibilities of extending our METTEL^2 tableau prover generation technology [23] to temporal logic. METTEL^2 presents a first step to implement the tableau calculus synthesis framework introduced in [18] and extends it to a tableau prover generation platform. The framework provides a theoretical foundation for sound, complete and terminating implementations of tableau procedures for a wide class of logics with first-order representable semantics and, in particular, for many logics which can be specified in METTEL^2. The method transforms the definition of a logic into a set of tableau rules that form a sound, complete and terminating tableau calculus for the logic, under certain conditions, if this is possible. Temporal logic is an interesting case study for the tableau synthesis endeavour because logics with eventualities such as temporal logics cannot be handled with standard, essentially first-order tableau approaches. In the current form the tableau calculus synthesis framework is not applicable to temporal logic or other logics with eventualities. In this paper we show however how the tableau prover generator MetTeL^2 can be used to generate an implemented prover for the tableau calculus introduced for TLC. Here we use the standard representation of the semantics for PTL which involves fix-point operators.

II. Temporal Logic with Cardinality Constraints

A TLC *problem* is a pair $(\varphi, \mathcal{C})$, where φ is a PTL formula and $\mathcal{C}$ is a finite set of *cardinality constraints*. The set of PTL formulae is defined as the smallest set such that all *propositions* are PTL formulae, and if φ and ψ are in PTL formulae, then so are **true**, $\neg\varphi, \varphi \vee \psi$, $\bigcirc\varphi, \varphi\,\mathcal{U}\,\psi$. A cardinality constraint $C^{\propto m}$ is a tuple $(C, \propto, m)$, where C is a set of literals (propositions and their negations), $\propto \in \{=, \leq\}$ and $m \in \mathbb{Z}$. The size $\#C$ of a literal set C is the number of literals in C.

A model for TLC problem $(\varphi, \mathcal{C})$ can be characterised as a sequence of *states* of the form $\sigma = s_0, s_1, s_2, s_3, \ldots$, where each state s_i is a set of propositional symbols representing those propositions, which are satisfied at the ith moment in time. We define the relations

$$\begin{array}{ll} \sigma \models C^{=m} & \text{iff for all } i \geq 0, \\ & \#\left((s_i \cap C) \cup \{\neg p \in C \mid p \notin s_i\}\right) = m \\ \sigma \models C^{\leq m} & \text{iff for all } i \geq 0, \\ & \#\left((s_i \cap C) \cup \{\neg p \in C \mid p \notin s_i\}\right) \leq m \end{array}$$

and

$$\begin{array}{lll} (\sigma, i) \models & p & \text{iff } p \in s_i \text{ where } p \text{ is a proposition} \\ (\sigma, i) \models & \neg\varphi & \text{iff it is not the case that } (\sigma, i) \models \varphi \\ (\sigma, i) \models & \varphi \vee \psi & \text{iff } (\sigma, i) \models \varphi \text{ or } (\sigma, i) \models \psi \\ (\sigma, i) \models & \bigcirc\varphi & \text{iff } (\sigma, i+1) \models \varphi \\ (\sigma, i) \models & \varphi\,\mathcal{U}\,\psi & \text{iff } \exists k \in \mathbb{N}.\ k \geq i \text{ and } (\sigma, k) \models \psi \text{ and} \\ & & \forall j \in \mathbb{N}, \text{ if } i \leq j < k \text{ then } (\sigma, j) \models \varphi \end{array}$$

Notice that the satisfiability of constraints does not depend on the time moment—constraints hold at every moment of time. The semantics of **true**, **false** and the other Boolean operators ($\wedge$, $\Rightarrow$, $\Leftrightarrow$) are as usual and we define $\Box$ (always in the future), $\Diamond$ (sometime in the future) and $\mathcal{W}$ (unless) as follows: $\Diamond\varphi = \textbf{true}\ \mathcal{U}\,\varphi$, $\Box\varphi = \neg\Diamond\neg\varphi$ and $\varphi\,\mathcal{W}\,\psi = (\varphi\,\mathcal{U}\,\psi) \vee (\Box\varphi)$. For a PTL formula φ and a set of constraints $\mathcal{C}$ we say that $(\varphi, \mathcal{C})$ is *satisfiable* iff there is some σ such that $(\sigma, 0) \models \varphi$ and $\sigma \models C$ for every $C \in \mathcal{C}$; for a set of PTL formulae $\mathcal{S}$, $(\mathcal{S}, \mathcal{C})$ is satisfiable iff $(\bigwedge_{\varphi \in \mathcal{S}}, \mathcal{C})$ is satisfiable.

Notice that the addition of constraints does not extend the logic, that is, we can rewrite any constraint $C^{\propto m}$ into the syntax of PTL, for example, as follows. For a constraint $C^{\propto m}$ define PTL formulae

$$\begin{array}{rcl} pos(C^{\propto m}) & = & \bigwedge_{\substack{U \subseteq C \\ |U| = |C|+1-m}} \Box \bigvee_{l_i \in U} l_i \\ neg(C^{\propto m}) & = & \bigwedge_{\substack{U \subseteq C \\ |U| = m+1}} \Box \bigvee_{l_i \in U} \neg l_i \end{array}$$

Then (it easily follows from [21]) that $\sigma \models C^{=m}$ iff $(\sigma, 0) \models pos(C^{=m}) \wedge neg(C^{=m})$ and $\sigma \models C^{\leq m}$ iff $(\sigma, 0) \models neg(C^{\leq m})$. For a constraint of the form $C^{\leq k}$, such a direct encoding contains $\binom{n}{k+1}$ clauses, which for $k = \lceil n/2 \rceil - 1$ reaches $O\left(2^n/\sqrt{n/2}\right)$ clauses [21]. This blow-up can be avoided by introducing extra propositions and using an n-bit counter to represent 2^n different values. For example, we could represent a constraint of the form $\{p_1, p_2, p_3, p_4\}^{=1}$ using just two propositional variables t'_1, t'_2 by adding a conjunction of these formulae.

$$\begin{array}{lclclcl} \Box(p_1 & \Leftrightarrow & (t'_1 \wedge t'_2)) & \quad & \Box(p_3 & \Leftrightarrow & (\neg t'_1 \wedge t'_2)) \\ \Box(p_2 & \Leftrightarrow & (t'_1 \wedge \neg t'_2)) & \quad & \Box(p_4 & \Leftrightarrow & (\neg t'_1 \wedge \neg t'_2)). \end{array}$$

Other translations based on different counter encodings can be found in the literature, see, for example, [2], [5], [21].

Our explicit representation of constraints, in contrast with the translations, not only is succinct but it also allows the prover to make use of this information in a lazy, on-the fly way. An evaluation of these approaches can be found in Section V.

III. The Tableau Calculus

In line with the type of tableau calculi generated in the framework of [18] and the kind of tableau calculi definable in the specification language of MetTeL^2 [23], our tableau calculus for TLC is a ground semantic tableau calculus operating on labelled expressions and equalities among labels and expressions.

First, we define *labels*. We introduce an additional function symbol f representing the *successor function* on labels. Then the set of all labels can be obtained by applying the following definitions inductively: (i) 0 is a label, and (ii) $f(\ell)$ is a label whenever ℓ is a label. For $i \geq 0$, $f^i(0)$ denotes the label representing the ith successor of 0, that is, $f^0(0) \overset{\text{def}}{=} 0$ and $f^{i+1}(0) \overset{\text{def}}{=} f(f^i(0))$.

Expressions in the language of the tableau calculus are TLC formulae defined with the additional binary connective $E_\Diamond$ and ternary connective $E_\mathcal{U}$ operating on PTL formulae. Expressions of the form $E_\Diamond(\ell, \varphi)$ and $E_\mathcal{U}(\ell, \varphi, \psi)$ are used to indicate (as yet) unfulfilled eventualities respectively of the kind $\Diamond\varphi$ and $\varphi\,\mathcal{U}\,\psi$ within a branch of a derivation. We refer to $E_\Diamond(\ell, \varphi)$ and $E_\mathcal{U}(\ell, \varphi, \psi)$ as *eventuality expressions*. Expressions in the tableau language are thus either PTL formulae, constraints, or eventuality expressions. A *labelled expression* has the form $\ell : \varphi$, where ℓ is a label and φ is an expression of the extended language.

Finally, *tableau formulae* are either labelled expressions or *equalities* of the form $\ell \approx \ell'$ or $\varphi \approx \psi$, where ℓ and ℓ' denote labels and φ and ψ denote expressions. The equalities trigger rewriting within branches, as described below.

In this paper a *tableau* or *tableau derivation* is a finitely branching, ordered tree whose nodes are sets of tableau formulae. Assuming that $\mathcal{S}$ and $\mathcal{C}$ are respectively the input sets of PTL formulae and constraints to be tested for satisfiability the root node of the tableau is the set $\{0 : \varphi \mid \varphi \in \mathcal{S}\} \cup \{0 : \Box C^{\propto m} \mid C^{\propto m} \in \mathcal{C}\}$. Successor nodes in the tableau are constructed through the application of the *inference rules* in the calculus. The inference rules have the general form $X_0/X_1 \mid \dots \mid X_n$, with X_i being sets of tableau formulae. X_0 is the set of premises and the X_i, for $1 \leq i \leq n$, are the sets of conclusions. If $n = 0$, the rule is called a *closure rule*, written $X_0/\bot$. An inference rule is applicable to a selected tableau formula E in a leaf node of the tableau, if E together with possibly other tableau formulae in the node, are simultaneous instantiations of all the premises of the rule. Then n successor nodes are created that contain the formulae of the current node and the appropriate instances of $X_1, \dots, X_n$. We call these successor nodes *derived nodes*.

As is standard, we assume that no rule is applied more than once to the same set of premises. Additionally, in order to avoid trivial redundancies we stipulate that a rule application to a leaf node $\mathcal{N}$ is *redundant* if there is an instantiated conclusion set X_i, for $1 \leq i \leq n$, of the rule application such that all formulae from X_i already appear in $\mathcal{N}$. In a tableau, a maximal path from the root node is called a *branch*.

Our calculus uses ordered rewriting to handle the equalities. The ordering $\prec$ we use is a lexicographical path ordering based on the order of appearance of variables, constants, and connectives during the inference process. Thus, for labels we have: $f^i(0) \prec f^j(0)$, if $i < j$. Note that, since eventuality expressions are not allowed to appear in the input set (and hence in the root node) and the way the rules are defined, it always holds that $\Diamond\varphi \prec E_\Diamond(\ell,\varphi)$ and $(\varphi \mathcal{U} \psi) \prec E_\mathcal{U}(\ell,\varphi,\psi)$. $\prec$ is a reduction ordering and, thus, a strict, total, and well-founded ordering on labels and expressions. The equalities on the branch form a ground rewrite system. Both forward and backward rewriting is used each time new conclusions are derived. In particular, if an equality of the form $f^i(0) \approx f^j(0)$ or $\psi \approx \varphi$ is added to a derived node, backward rewriting is applied. This means the rewrite system is rebuilt with respect to the newly added equality, and all formulae of the node are rewritten with respect to the rewrite system. Forward rewriting with respect to the current rewrite system is applied to all newly derived nodes during the derivation.

For a branch $\mathcal{B}$ of a tableau we write $E \in \mathcal{B}$ to indicate that the tableau formula E has been derived and *persists* in $\mathcal{B}$, that is, for some node $\mathcal{N}$ of the branch $\mathcal{B}$, E belongs to $\mathcal{N}$ and every node that is a descendant from $\mathcal{N}$. This means that E is not rewritten in $\mathcal{N}$ and all subsequent nodes. We say that E *appears in a node of* $\mathcal{B}$ if E belongs to this node or can subsequently be rewritten in $\mathcal{B}$.

The tableau rules of our calculus for TLC are listed in Figure 1. The rules for the temporal operators, apart from $(\Diamond)$ and $(\mathcal{U})$, are standard decomposition rules for PTL formulae. These decompose formulae into subformulae that must hold now and, where necessary, in the next moment. The $(\Diamond)$-rule is not the usual diamond rule, but replaces $\Diamond\varphi$ by $E_\Diamond(\ell,\varphi)$ indicating that the eventuality $\Diamond\varphi$ is not yet fulfilled. Until eventualities are handled in a similar way to diamond eventualities. Note that there is no rule for negated $\mathcal{W}$ formulae since we assume that the input does not contain any negated occurrences of $\mathcal{W}$.

Rules for temporal operators:

$$\frac{\ell : \neg\neg\varphi}{\ell : \varphi}(\neg\neg) \qquad \frac{\ell : \varphi \vee \psi}{\ell : \varphi \mid \ell : \psi}(\vee) \qquad \frac{\ell : \neg(\varphi \vee \psi)}{\ell : \neg\varphi, \ell : \neg\psi}(\neg\vee)$$

$$\frac{\ell : \bigcirc\varphi}{f(\ell) : \varphi}(\bigcirc) \qquad \frac{\ell : \neg\bigcirc\varphi}{\ell : \bigcirc\neg\varphi}(\neg\bigcirc)$$

$$\frac{\ell : \Diamond\varphi}{\ell : E_\Diamond(\ell,\varphi)}(\Diamond) \qquad \frac{\ell : \neg\Diamond\varphi}{\ell : \Box\neg\varphi}(\neg\Diamond)$$

$$\frac{\ell : \Box\varphi}{\ell : \varphi, f(\ell) : \Box\varphi}(\Box) \qquad \frac{\ell : \neg\Box\varphi}{\ell : \Diamond\neg\varphi}(\neg\Box)$$

$$\frac{\ell : \varphi\mathcal{U}\psi}{\ell : E_\mathcal{U}(\ell,\varphi,\psi)}(\mathcal{U}) \qquad \frac{\ell : \neg(\varphi\mathcal{U}\psi)}{\ell : \neg\psi\,\mathcal{W}\,\neg(\varphi \vee \psi)}(\neg\mathcal{U})$$

$$\frac{\ell : \varphi\mathcal{W}\psi}{\ell : \psi \mid \ell : \varphi, f(\ell) : \varphi\mathcal{W}\psi}(\mathcal{W})$$

Rules for constraints:

$$\frac{\ell : (\{p\} \cup C)^{\propto m}, \ell : \neg p}{\ell : C^{\propto m}}(C^+_-) \qquad \frac{\ell : (\{p\} \cup C)^{\propto m}, \ell : p}{\ell : C^{\propto (m-1)}}(C^+_+)$$

$$\frac{\ell : (\{\neg p\} \cup C)^{\propto m}, \ell : p}{\ell : C^{\propto m}}(C^-_+) \qquad \frac{\ell : (\{\neg p\} \cup C)^{\propto m}, \ell : \neg p}{\ell : C^{\propto (m-1)}}(C^-_-)$$

$$\frac{\ell : (\{p\} \cup C)^{\propto m}}{\ell : p \mid \ell : \neg p}(\mathsf{cut}^+) \qquad \frac{\ell : (\{\neg p\} \cup C)^{\propto m}}{\ell : p \mid \ell : \neg p}(\mathsf{cut}^-)$$

Rules for fulfilling eventualities:

$$\frac{\ell : E_\Diamond(\ell',\varphi)}{\ell : \varphi, E_\Diamond(\ell',\varphi) \approx \Diamond\varphi \mid f(\ell) : E_\Diamond(\ell',\varphi)}(E_\Diamond)$$

$$\frac{\ell : E_\mathcal{U}(\ell',\varphi,\psi)}{\ell : \psi, E_\mathcal{U}(\ell',\varphi,\psi) \approx (\varphi\mathcal{U}\psi) \mid \ell : \varphi, f(\ell) : E_\mathcal{U}(\ell',\varphi,\psi)}(E_\mathcal{U})$$

Closure rules:

$$\frac{\ell : \varphi, \ell : \neg\varphi}{\bot}(\mathsf{clash}) \qquad \frac{\ell : C^{\propto -1}}{\bot}(\mathsf{empty})$$

$$\frac{\ell : C^{=m}, \#C < m}{\bot}(\mathsf{cap})$$

$$\frac{\ell : E_\Diamond(\ell',\varphi)}{\bot}(E_\Diamond\text{-}\mathsf{test}) \qquad \frac{\ell : E_\mathcal{U}(\ell',\varphi,\psi)}{\bot}(E_\mathcal{U}\text{-}\mathsf{test})$$

Unrestricted blocking rule:

$$\frac{\ell : \varphi, \ell' : \psi}{\ell \approx \ell' \mid \ell \not\approx \ell'}(\mathsf{ub})$$

Fig. 1. Tableau rules for TLC

To understand the rules for the constraints suppose that $\propto$ denotes $=$. The intuition of the rule (C^+_-) is as follows: if $(\{p\} \cup C)^{=m}$ is true and $\neg p$ is true, that is, p is not true, then $C^{=m}$ must be true. The rule (C^+_+) says that if $(\{p\} \cup C)^{=m}$ and p are true then $C^{=(m-1)}$ must be true. The rules (C^-_+) and (C^-_-) are duals. The intuitions are similar for $\propto$ denoting $\leq$. The (cut^+) and (cut^-)-rules are DPLL type analytic cut rules enumerating the possible truth assignments to propositional symbols occurring in constraints.

The $(E_\Diamond)$-rule tries to fulfil a $\Diamond$ eventuality or delegate fulfilment to the next moment of time. Note that in the left derived node when an eventuality is fulfilled the equality $E_\Diamond(\ell,\varphi) \approx \Diamond\varphi$ is added as well. This triggers rewriting of $E_\Diamond(\ell,\varphi)$ to $\Diamond\varphi$ in this node. Then there are no more occurrences of $E_\Diamond(\ell,\varphi)$ in the node and means that the eventuality $\Diamond\varphi$ is fulfilled. The idea of the $(E_\mathcal{U})$-rule is the same as for the $(E_\Diamond)$-rule.

The calculus includes the standard closure rule (clash), two closure rules for constraints and two closure rules for eventualities. The rule (empty) closes a branch for a constraint $C^{=-1}$ or $C^{\leq -1}$, because constraints cannot contain a negative number of literals. The (cap)-rule is an early closure rule for the case that m literals of a constraint must hold but the constraint already contains less than m literals. The rules ($E_{\Diamond}$-test) and ($E_{\mathcal{U}}$-test) close a branch when the eventualities are not fulfilled. Here it is clear that without an appropriate strategy for rule application the calculus is not sound.

The (ub)-rule is the unrestricted blocking rule introduced in [17] for deciding expressive description logics with full role negation. It conjectures the equality of labels occurring in a leaf node. Through rewriting the time points ℓ and ℓ' are joined in the left derived node. If no models can be found by continuing the derivation, that is, the sub-tableau from this point onward is closed, then the right derived node is expanded further. Combined with the rule application strategy we define next it performs a form of ancestor blocking.

The calculus is sound and complete if the priority of the rules from highest to lowest are:

$$
\begin{array}{l}
\text{(clash), (empty), (cap)} \\
\quad > \ (\neg\neg), (\neg\vee), (\neg\bigcirc), (\Diamond), (\neg\Diamond), (\neg\,\Box), (\,\mathcal{U}\,), (\neg\,\mathcal{U}\,) \\
\quad > \ (C^{+}_{-}), (C^{+}_{+}), (C^{-}_{+}), (C^{-}_{-}) \ > \ (\vee) \ > \ (\text{cut}^{+}), (\text{cut}^{-}) \\
\quad > \ \text{(ub)} \ > \ (\bigcirc), (\,\Box), (\,\mathcal{W}\,), (E_{\Diamond}), (E_{\mathcal{U}}) \ > \ (E_{\Diamond}\text{-test}), (E_{\mathcal{U}}\text{-test})
\end{array}
$$

This is the rule application priority used in our implementation (described in Section V), but there is in fact more flexibility. What is crucial is that the ($E_{\Diamond}$-test) and ($E_{\mathcal{U}}$-test) rules are only applied if no other rules are applicable, because then the branch cannot be continued to fulfil the eventualities. Further, any application of a label-introducing rule (that is, a δ-rule), namely the rules ($\bigcirc$), ($\Box$), ($\mathcal{W}$), ($E_{\Diamond}$) and ($E_{\mathcal{U}}$), must be delayed until after all local rules have been applied. The local rules are the rules: (clash), (empty), (cap), ($\neg\neg$), ($\neg\vee$), ($\neg\bigcirc$), ($\Diamond$), ($\neg\Diamond$), ($\neg\,\Box$), ($\mathcal{U}$), ($\neg\mathcal{U}$), (C^{+}_{-}), (C^{+}_{+}), (C^{-}_{+}), (C^{-}_{-}), ($\vee$), (cut^{+}) and (cut^{-}). It is sensible to delay the application of the (ub)-rule until the local rules have been applied to a label, because then all labels are 'maximally expanded'.

Additionally, in order to obtain termination of the calculus, we assume that the following *avoid huge branch (AHB)* derivation strategy (cf. [19]) is used in the tableau derivations: Limit the number of different labels in every node to $2^{n+1} + 4^{n+1}$, where n is the length of the input, and discard branches where the leaf node contains more labels. $2^{n+1} + 4^{n+1}$ is the model bound established in the effective finite model property result below (Lemma 1).

We use the notation $T_{\mathsf{TLC}}(\mathcal{S},\mathcal{C})$ for a maximally expanded tableau (using the AHB strategy) built by applying the rules of the calculus T_{TLC} starting with a set $\mathcal{S}$ of PTL formulae and a constraint set $\mathcal{C}$ as input. As usual we assume that all the rules of the calculus are applied non-deterministically to a tableau within the constraints of the rule application strategy. A branch of a tableau is *closed* if a closure rule has been applied in this branch, otherwise the branch is called *open*. The tableau $T_{\mathsf{TLC}}(\mathcal{S},\mathcal{C})$ is *closed* if all its branches are closed or discarded and $T_{\mathsf{TLC}}(\mathcal{S},\mathcal{C})$ is *open* otherwise. The calculus T_{TLC} is *sound* iff each $T_{\mathsf{TLC}}(\mathcal{S},\mathcal{C})$ is open whenever $(\mathcal{S},\mathcal{C})$ is satisfiable. T_{TLC} is *complete* iff for any unsatisfiable $(\mathcal{S},\mathcal{C})$ there is a closed $T_{\mathsf{TLC}}(\mathcal{S},\mathcal{C})$. T_{TLC} is said to be *terminating* (for satisfiability) iff for every *finite* $(\mathcal{S},\mathcal{C})$ every tableau $T_{\mathsf{TLC}}(\mathcal{S},\mathcal{C})$ is finite.

Next we show the application of the tableau calculus to an example. Here, we consider a tableau $T_{\mathsf{TLC}}(\mathcal{S},\mathcal{C})$ for the set of formulae $\mathcal{S} = \{\,\Box\neg p,\ \Box q,\ \Box r, \Diamond r\}$ and the set of constraints $\mathcal{C} = \{\{p,q\}^{=1}, \{q,r,s\}^{=2}\}$. This TLC problem is satisfiable and a model is, for example, $s_i = \{q,r\}$ for all $i \in \mathbf{N}$. The tableau derivation is shown in Figure 2. As our tableau rules are accumulating we write the contents of the nodes in a tableau derivation in the order in which the formulae are derived (where a depth-first left-to-right search strategy is used). Each line in the figure is numbered on the left. The rule applied and the number of the premise(s) to which it was applied to produce the labelled expression in each line are specified on the right. The black triangles denote branching points in the derivation. A branch expansion after a branching point is indicated by appropriate indentation.

Note that the lines 12, 14, 16, 20 are redundant as $f(0)$ rewrites to 0. Also note in the open branch after line 26 applications of the rules (cut^{+}) and (cut^{-}) are redundant and not performed. Similarly, line 29 is redundant because using forward rewriting $f(0)$ rewrites 0 and $0 : E_{\Diamond}(0,r)$ is already in the branch. Further, while the application of the (ub)-rule at step 10 allows to find a maximally expanded open branch quickly by identifying labels $f(0)$ and 0 the derivation on the right branch with $0 \not\approx f(0)$ can potentially be infinite if the AHB strategy is not used; for instance, if we change $\Box r$ to $\Box\neg r$ in the example. In this case $\Diamond r$ would never be fulfilled and the tableau would not terminate.

IV. Soundness, Completeness and Termination

Next we address soundness, completeness and termination of the calculus. Missing proofs can be found in the technical report [10].

We extend the notion of satisfiability to the additional constructs of the tableau language:

$$
\begin{array}{llll}
(\sigma,i) \models & f^{j}(0) : \varphi & \text{iff} & (\sigma,j) \models \varphi \\
(\sigma,i) \models & E_{\Diamond}(\ell,\varphi) & \text{iff} & (\sigma,i) \models \ell : \Diamond\varphi \\
(\sigma,i) \models & E_{\mathcal{U}}(\ell,\varphi,\psi) & \text{iff} & (\sigma,i) \models \ell : (\varphi\,\mathcal{U}\,\psi)
\end{array}
$$

As a consequence, rewriting of $E_{\Diamond}(\ell,\varphi)$ to $\Diamond\varphi$ and $E_{\mathcal{U}}(\ell,\varphi,\psi)$ to $\varphi\,\mathcal{U}\,\psi$ in a satisfiable tableau node cannot change satisfiability of the node.

Given a set of TLC-formulae $\mathcal{S}$ and a set of constraints $\mathcal{C}$ we define the *length* of the input $(\mathcal{S},\mathcal{C})$ as the sum of lengths of all formulae in $\mathcal{S}$ in symbols plus the number of literals in all constraints from $\mathcal{C}$. A sequence of states $\sigma = s_0, s_1, s_2, s_3, \ldots$ is *ultimately periodic* [22] with index i and period m if for all $k \geq i$ we have $s_k = s_{k+m}$. Notice that a TLC-problem $(\mathcal{S},\mathcal{C})$ has the same models as the the union of $\mathcal{S}$ and a representation

1.	$0 : \Box\neg p$	given
2.	$0 : \Box q$	given
3.	$0 : \Box r$	given
4.	$0 : \Diamond r$	given
5.	$0 : \Box\{p, q\}^{=1}$	given
6.	$0 : \Box\{q, r, s\}^{=2}$	given
7.	$0 : E_{\Diamond}(0, r)$	$(\Diamond)$,4
8.	$0 : \neg p$	$(\Box)$,1
9.	$f(0) : \Box\neg p$	$(\Box)$,1
10.	$\blacktriangleright 0 \approx f(0)$	(ub),1,9
		$f(0)$ is/will be rewritten to 0
11.	$0 : q$	$(\Box)$,2
12.	$f(0) : \Box q$	$(\Box)$,2
13.	$0 : r$	$(\Box)$,3
14.	$f(0) : \Box r$	$(\Box)$,3
15.	$0 : \{p, q\}^{=1}$	$(\Box)$,5
16.	$f(0) : \Box\{p, q\}^{=1}$	$(\Box)$,5
17.	$0 : \{q\}^{=1}$	(C^{+}_{-}),15,8
18.	$0 : \{\}^{=0}$	(C^{+}_{+}),17,11
19.	$0 : \{q, r, s\}^{=2}$	$(\Box)$,6
20.	$f(0) : \Box\{q, r, s\}^{=2}$	$(\Box)$,6
21.	$0 : \{r, s\}^{=1}$	(C^{+}_{+}),19,11
22.	$0 : \{s\}^{=0}$	(C^{+}_{+}),21,13
23.	$\blacktriangleright 0 : s$	(cut^{+}),22
24.	$0 : \{\}^{=-1}$	(C^{+}_{+}),22,23
	Closed	(empty),24
25.	$\blacktriangleright 0 : \neg s$	(cut^{+}),22
26.	$0 : \{\}^{=0}$	(C^{+}_{-}),22,25
27.	$\blacktriangleright 0 : r$	$(E_{\Diamond})$,7
28.	$E_{\Diamond}(0, r) \approx \Diamond r$	$(E_{\Diamond})$,7
		$E_{\Diamond}(0, r)$ is rewritten to $\Diamond r$
	Maximally expanded, open	
29.	$\blacktriangleright f(0) : E_{\Diamond}(0, r)$	$(E_{\Diamond})$,7
		redundant as $f(0)$ rewrites to 0
30.	Closed	$(E_{\Diamond}$-test),7
31.	$\blacktriangleright 0 \not\approx f(0)$	(ub),1,9
$\cdots$		

Fig. 2. Example of derivation in TLC

of constraints as PTL formula. Therefore, Theorem 4.7 in [22] has the following corollary.

Lemma 1 (Effective Finite Model Property): Let $\mathcal{S}$ be a set of TLC-formulae, $\mathcal{C}$ be a set of constraints and n be the length of the input $(\mathcal{S}, \mathcal{C})$. $\mathcal{S}$ is satisfiable with respect to $\mathcal{C}$ iff $\mathcal{S}$ is satisfiable with respect to $\mathcal{C}$ in an ultimately periodic model with $i + m \leq 2^{n+1} + 4^{n+1}$.

It is not difficult to see that all the rules of the TLC tableau calculus except the rules ($E_{\Diamond}$-test) and ($E_{\mathcal{U}}$-test) preserve satisfiability. Using Lemma 1, the following can be proved similarly to Theorem 14 in [19].

Theorem 1: Let $\mathcal{S}$ be a set of TLC-formulae which is satisfiable with respect to a set of constraints $\mathcal{C}$ and n be the length of $(\mathcal{S}, \mathcal{C})$. Then in every T_{TLC} tableau for the input $(\mathcal{S}, \mathcal{C})$, there exists an open branch $\mathcal{B}$ such that the number of persistent labels in $\mathcal{B}$ does not exceed $2^{n+1} + 4^{n+1}$.

As a corollary, soundness of the tableau calculus follows.

Theorem 2 (Soundness): T_{TLC} is sound, that is $T_{\mathsf{TLC}}(\mathcal{S}, \mathcal{C})$ is open for every satisfiable input $(\mathcal{S}, \mathcal{C})$.

Inspecting all the tableau rules in T_{TLC} further, it is not difficult to show that the number of (unlabelled) expressions which can appear under labels in a tableau $T_{\mathsf{TLC}}(\mathcal{S}, \mathcal{C})$ is limited by an exponential function in the size of the input $(\mathcal{S}, \mathcal{C})$. Since all the branches with more than $2^{n+1} + 4^{n+1}$ labels are discarded according to the AHB strategy, the number of all labelled formulae in every branch of a tableau can be limited and, consequently, the number of rule applications in every branch can be limited, too. As a corollary, we obtain termination of the calculus.

Theorem 3 (Termination): T_{TLC} is a terminating tableau calculus for satisfiability in TLC if the AHB strategy is used.

Now we concentrate on proving completeness. Let $\mathcal{B}$ be a maximally expanded open branch in T_{TLC}-tableau. We construct an (ultimately periodic) TLC-model $\sigma_{\mathcal{B}}$ from the open branch as follows. Recall that all the branches in the tableau derivation are finite (if a branch becomes too long, it is discarded according to the AHB strategy). Let $0, f(0), \ldots, f^{n-1}(0)$ be all the labels which occur in persistent formulae of $\mathcal{B}$ (that is, they are never rewritten to smaller labels). Thus, $f^{n}(0)$ have been rewritten in $\mathcal{B}$ to some $f^{k}(0)$ with some k in $\{0, \ldots, n-1\}$. Let states of the model be defined as follows:

$$s_i \overset{\text{def}}{=} \begin{cases} \{p \mid f^{i}(0) : p \in \mathcal{B}\}, & \text{if } i \in \{0, \ldots, n-1\}, \\ s_{k+[(i-k) \pmod{n-k}]}, & \text{otherwise.} \end{cases}$$

Lemma 2: $(\sigma_{\mathcal{B}}, i) \models \varphi$ for every persistent labelled formula $f^{i}(0) : \varphi \in \mathcal{B}$.

A direct consequence of this lemma is the Completeness Theorem.

Theorem 4 (Completeness): T_{TLC} is complete tableau calculus for TLC. That is, if $T_{\mathsf{TLC}}(\mathcal{S}, \mathcal{C})$ is open then $\mathcal{S}$ is satisfiable in an (ultimately periodic) TLC-model with respect to the set of constraints $\mathcal{C}$.

V. Implementation and Experiments

We have implemented the tableau calculus described in Section III as a prover, called LTLC. The LTLC prover has been built using the METTEL^2 automated prover generator technology. The input of the METTEL^2 system is the definition of the syntax of a logic or logical theory plus the definition of a tableau calculus within the specified syntax. When run, it attempts to generate automatically the JAVA code of a prover for the specified tableau calculus. JAVA was chosen as it is a widely known object-oriented programming language and, as such, enables the easy modification of generated provers. The

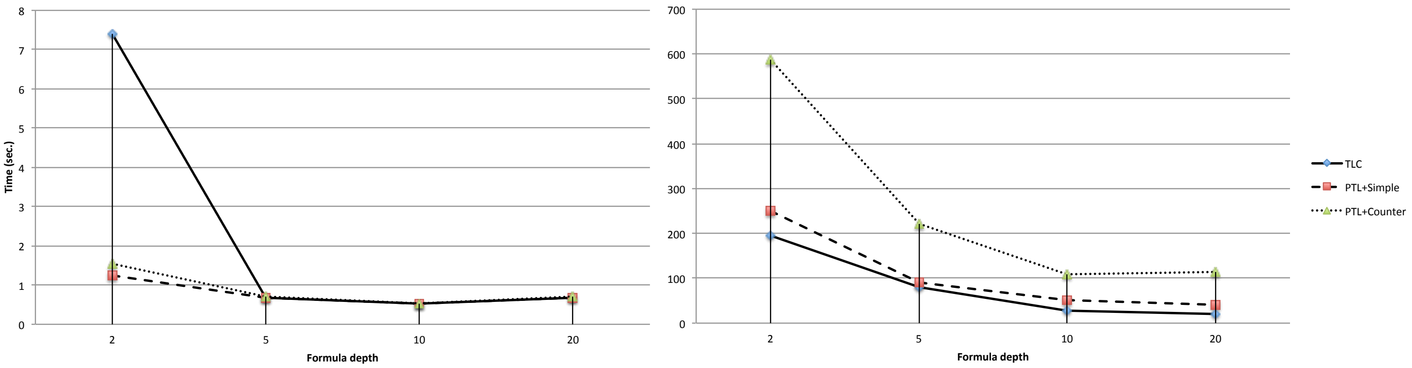

Fig. 3. Problems with 1 constraint (left) and 5 constraints (right)

code for LTLC was generated automatically with MetTeL^2 from the definition of the TLC syntax and the tableau rules presented in Section III. For the implementation we used several additional tableau rules not necessary for completeness of the calculus. In particular, in order to improve performance the rule specification for the implementation includes unit propagation rules. The reasoning core of MetTeL^2 also fully supports dynamic backtracking and backjumping techniques which are commonly used to reduce search space in tableau derivations. Consequently, both the techniques are available in LTLC. LTLC uses depth-first left-to-right search selection strategy which is the default in MetTeL^2.

In the implementation, we have limited the language of constraints to constraints of the form $C^{=m}$. Since the specification languages of MetTeL^2 do not yet support connectives with variable arity and schematic rules, for the current prototype implementation the total number of literals that can occur in a constraint is at most 4. Then, constraints can be represented by a finite set of connectives $C_{=,m,n}$ for $0 \leq m \leq n$ and $2 \leq n \leq 4$. For example, the constraint $\{x, y, z, w\}^{=1}$ is represented as $C_{=,1,4}(x, y, z, w)$, where $C_{=,1,4}$ is a connective of arity 4 specifying that at most 1 out of its 4 arguments is true at time point ℓ (here, x, y, z, w denote literals). The connectives $C_{=,m,n}$ for $n < 2$ are omitted because $C_{=,1,1}(x)$ is equivalent to x, $C_{=,0,1}(x)$ is equivalent to $\neg x$, $C_{=,0,0}$ is equivalent to **true**, and $C_{=,-1,n}(x_1, \ldots, x_n)$ is equivalent to **false** for $0 \leq n \leq 4$. In addition, appropriate instances of the constraint tableau rules plus the (empty) and (cap) rules in the calculus need to be added. For example, the tableau calculus rule (C^{+}_{-}) for $C^{=1}$ is instantiated as three tableau rules for the constraints of the lengths 2, 3 and 4. The instance of this rule for the constraint $C_{=,1,4}$ is as follows (below top).

$$\frac{\ell : C_{=,1,4}(p, x_1, x_2, x_3), \ell : \neg p}{\ell : C_{=,1,3}(x_1, x_2, x_3)}$$

$$\frac{\ell : C_{=,1,n}(x_1, x_2, \ldots, x_n)}{\ell : C_{=,1,n}(x_2, \ldots, x_n, x_1)}$$

In order to simulate the property that every constraint $C^{\propto m}$ is an unordered set we generate permutations of arguments of $C_{=m,n}$ by the three rules for $n = 2, 3, 4$ shown above (bottom). These rules allow the above rule for constraint reduction to be applied to p regardless of in which position it can be found in the original constraint. Similar instantiations take place for other constraint rules in the calculus.

MetTeL^2 is a prototypical system intended for experimenting with tableau calculi and prover generation for various logics and we do not expect the LTLC prover to compete with the highly optimised state-of-the-art temporal tableau-based or resolution-based provers. In order to check the feasibility of our approach to constraint reasoning, we have applied LTLC to randomly generated problems and to their translations to PTL. Then the same reasoning engine is applied to the TLC and PTL formulae, which allows us to see relative advantages and disadvantages of a constraint language and an associated calculus compared with constraints being expressed as PTL formulae.

Because constraints are approximated in the current version of MetTeL^2, in all our experiments the constraints were "exactly one out of four literals is true at any time"; however, one input problem can contain a number, m, of such constraints. The PTL part of the input was randomly generated as follows: first a tree of specified height, n, was generated with every internal node having one or two successors with equal probability. Then unary (or binary) temporal and Boolean operators were assigned equi-probably to nodes with one (respectively, two) successors. The literals in the formula leaves and in the constraints were randomly drawn from the set of size 20. We have generated 100 random problems for every set of values (m, n).

All experiments were conducted on a PC equipped with an Intel 2.93GHz Core i7 CPU and 4GiB of main memory running under MacOS X 10.6.8. Every computation was restricted to a 600 second time limit and a 1GiB memory limit. We compared the performance of LTLC on a given TLC problem and on its two translations defined in Section II: the "simple" translation and the translation using a two-bit counter to determine which one of the four literals is true. Any computation that did not successfully terminate within the specified limits was recorded as a timeout (600 seconds).

The performance of LTLC on problems with one and five constraints and the formula depths varying between 2 and 20

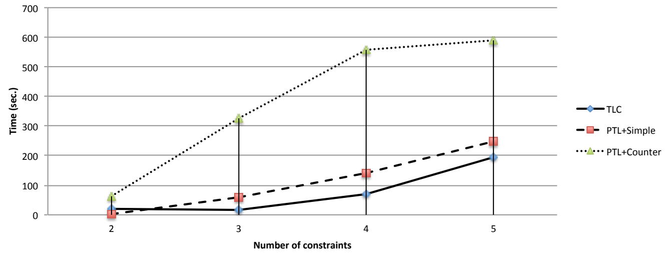

Fig. 4. Problems with a varying number of constraints and depth 2 formulae

is given in Fig. 3. Interestingly, in both cases the running time decreases as the formula depth grows. This is caused by the growing proportion of unsatisfiable problems in the input: all problems with formulae of depth 2 are satisfiable, while only 30% of problems with formulae of depth 5 and 20% of problems of depth 10 and 20 are satisfiable. For unsatisfiable problems the prover does not have to explore all possible tableau branches so the the running time is smaller.

As the graphs show, when the formula depth and the number of constraints are small the overheads of working with explicit constraints degraded the performance of the system; this discrepancy diminished as the formula depth grew. In case of 5 constraints, constraint reasoning outperformed both simple and counter-based constraint representation.

Finally, in Fig. 4 we give the running times of LTLC on formulae of depth 2 with the number of constraints varying between 2 and 5. It can be seen that reasoning in the presence of explicit constraints was faster as compared with either translation to PTL, especially as the number of constraints grew.

The above results show that the LTLC prover outperformed the 'simple' translation and the 'counter' translation for three constraints for the formula depths tried. Figure 3 shows that it is not worth using LTLC on problems with a low number of constraints which confirms earlier results from the implementation related to [8]. Regarding the 'counter' translation this performed much worse than the other two cases in Figure 3 (right) and Figure 4. We believe this to be because it uses $\Leftrightarrow$ and generates many $\Box$-formulae which slowed down the prover. While the results for the 'simple' translation Figure 3 (right) and Figure 4 were better than for the 'counter' translation we expect the performance of the prover on problems using the 'simple' translation to degrade with longer constraints. Due to restrictions in this version of METTEL^2 we are limited to constraints with a maximum length of 4 so could not test this hypothesis. Overall we believe that the results show the potential of this logic and calculus.

VI. Discussion and Related Work

The tableau method for PTL was originally introduced by [24] and implemented in several systems, for example, the Logics Workbench [3] based on the algorithms in [15], [20], the Tableau Workbench [1], those of Widmann and Goré based on [20], [24], Goranko et al. [14] based on [24] and TTM [13]. In this paper we used METTEL^2 to provide an implementation for our TLC calculus. Alternatively we could have considered using the Tableau Workbench [1] or the LoTReC system [11] which both allow the generation of implementations of user-defined tableau calculi as well as providing pre-defined tableau implementations for several logics. However, our tableau calculus may have had to be designed differently to satisfy the requirements of these systems.

Since TLC has fix-point operators, the tableau synthesis method from [18] can not be straightforwardly applied to TLC. That is why we designed the calculus and proved its soundness and completeness directly. However, we are interested in extending the tableau synthesis framework of [18] to fix-point logics with guaranteed soundness and completeness of the synthesised calculi for such logics. LTL and TLC constitute good case studies for research in this direction. Because METTEL^2 was initially designed to provide prover generation for the logics covered by the tableau synthesis framework, the logics outside the framework such as LTL and TLC are also attractive case studies for testing the flexibility of the METTEL^2 specification languages and the prover generation technology which is implemented in METTEL^2.

Following a traditional tableau approach for modal and description logics we choose to design the TLC calculus for METTEL^2 as a calculus in which the rules operate on labelled expressions. The search strategy required for the soundness, completeness and termination of the calculus means that there are some similarities to state-less tableau calculi operating on sets of formulae. However our tableau calculus and implementation is different to any other approaches and implementations. Most other tableau provers for PTL use equality blocking (usually implicitly). Our blocking is realised through the unrestricted blocking rule [17] and equality reasoning based on ordered rewriting. For PTL and TLC, because the constructed model is based on a linear sequence of states, this amounts to doing ancestor blocking. The models constructed with our method tend to have smaller domains.

Our technique to handle eventualities is inspired by algorithms which are commonly used for calculating fix-points in the μ-calculus [16]. Bad loops and good loops in the tableau derivation are identified by checking that all eventualities in the current branch are fulfilled. A somewhat similar technique is used in [6] for propositional dynamic logic but our technique is different in various ways. We discuss the diamond operator below but until eventualities are handled similarly. We extend the logical language with connectives, $E_\Diamond$ and $E_\mathcal{U}$, and introduce the rules $(E_\Diamond)$ and $(E_\mathcal{U})$ for unravelling the eventualities. It is important that in contrast to [6] the left conclusions of these rules contain equality expressions and since METTEL^2 supports rewriting of arbitrary (ground) expressions the rules trigger rewriting in the left derived nodes. In this case, the particular eventuality expression $E_\Diamond(\ell, \varphi)$, is rewritten and disappears from the node indicating that eventuality $\Diamond\varphi$, is fulfilled for the label ℓ. In the right derived node the rule $(E_\Diamond)$ leaves the expression $E_\Diamond(\ell, \varphi)$, untouched indicating that the eventuality is not fulfilled yet. As the other introduced rule ($E_\Diamond$-test) is applied only when no other rules are applicable and the branch is not already too long, these rules close the branch in the case that some eventuality is still not fulfilled in the branch.

Further difficulty in devising the TLC tableau calculus for generating a prover in MetTeL^2 is that unsatisfiable branches of tableau derivations using the presented calculus are not necessarily finite. The problem is caused by an interaction of unfulfilled eventualities and the unrestricted blocking mechanism used in MetTeL^2. In fact, while the left branches of the (ub)-decision points are closed if there are unfulfilled eventualities, the right branches can be infinite. In order to solve this problem we use the AHB strategy which stops expansion of a branch after it reaches some threshold. There are two potential improvements here. One is to calculate a more precise bound for this threshold and the other is to adopt standard loop-checking mechanisms.

This paper uses the same logic, TLC, as defined in [8]. That paper considered the complexity of TLC but checked the satisfiability of TLC formulae using an explicit model construction algorithm. In [9] we defined a resolution calculus for a related logic, called TLX, where the constraints are of the form: "exactly one" of a set of literals must hold, and no literal (or its negation) can appear in more than one set of constraints. Hence TLX is more restrictive than TLC.

VII. Conclusions

In this paper we have introduced a tableau calculus for reasoning in the logic TLC, which is propositional linear time temporal logic with additional constraints. The calculus is shown to be sound, complete and terminating. A prototype implementation of the calculus has been developed using MetTeL^2, an automated tableau prover generator. As MetTeL^2 currently does not support connectives with varying arity, the length of the constraints has been fixed as at most 4. We have experimented with the implementation comparing randomly generated problems in the TLC calculus with the same problems where the constraints are translated into PTL formulae in two different ways. The results show that the generated LTLC prover outperforms the translated problems where the number of constraints is three or above in all the cases we have tried. The work also provides a useful case study for extending the tableau synthesis technologies to handle temporal logics.

We would like to extend this work as follows. First, we need to extend MetTeL^2 to deal with constraints in a better way, for example, by allowing constraint rules with arbitrary arity. Additionally, rather than using the cut rules (cut$^+$) and (cut$^-$) we could replace these by a local satisfiability check of the propositional formulae within some label which could be performed by some external, fast, propositional solver such as a SAT solver. We could also explore whether a state-less, one pass-style implementation of the temporal part (like that in [20]) would speed up the temporal part of the prover.

Finally, the tableau rules can potentially be modified to allow for *dynamic* changes of the set of constraints. This allows us to accommodate constraints *into* the language as a logical connective. So as well as stating that the constraints hold at all moments one would be able to express constraints stating, for example, that "*eventually* exactly (or at most) l literals will be satisfied".

Acknowledgements. The authors were partially supported by EPSRC grants EP/D060451 (Dixon), EP/H043594/1 (Konev) and EP/H043748/1 (Schmidt, Tishkovsky).

References

[1] P. Abate and R. Gore. The tableaux work bench. In *Tableaux'03*, vol. 2796 of *LNAI*, pp. 230–236. Springer, 2003.

[2] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *CP 2003*, vol. 2833 of *LNCS*, pp. 108–122. Springer, 2003.

[3] P. Balsiger, A. Heuerding, and S. Schwendimann. The Logics Workbench 1.0. In *Tableaux'98*, vol. 1397 of *LNCS*, pp. 35–37. Springer, 1998.

[4] H. Barringer, M. Fisher, D. Gabbay, and G. Gough, eds. *Advances in Temporal Logic*, vol. 16 of *Applied Logic Series*. Kluwer, 2000.

[5] B. Benhamou, L. Sais, and P. Siegel. Two proof procedures for a cardinality based language in propositional calculus. In *STACS'94*, vol. 775 of *LNCS*, pp. 71–82. Springer, 1994.

[6] G. De Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for Converse-PDL. *Information and Computation*, 162:117–137, 2000.

[7] S. Demri and P. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1):84–103, 2002.

[8] C. Dixon, M. Fisher, and B. Konev. Temporal logic with capacity constraints. In *FroCoS'07*, vol. 4720 of *LNCS*, pp. 163–177. Springer, 2007.

[9] C. Dixon, M. Fisher, and B. Konev. Tractable temporal reasoning. In *IJCAI'07*, pp. 318–323, 2007.

[10] C. Dixon, B. Konev, R. A. Schmidt, and D. Tishkovsky. Labelled tableaux for temporal logic with cardinality constraints. Available at www.mettel-prover.org/papers/dkst12.pdf.

[11] L. Fariñas del Cerro, D. Fauthoux, O. Gasquet, A. Herzig, D. Longin, and F. Massacci. Lotrec: the generic tableau prover for modal and description logics. In *IJCAR'01*, vol. 2083 of *LNAI*, pp. 453–458. Springer, 2001.

[12] M. Fisher, D. Gabbay, and L. Vila, eds. *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier, 2005.

[13] J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. Dual Systems of Tableaux and Sequents for PLTL. *Journal of Logic and Algebraic Programming*, 78:701–722, 2009.

[14] V. Goranko, A. Kyrilov, and D. Shkatov. Tableau tool for testing satisfiability in LTL: Implementation and experimental analysis. *Electronic Notes in Theoretical Computer Science*, 262:113–125, 2010.

[15] G. L. J. M. Janssen. *Logics for digital circuit verification—theory, algorithms, and applications*. PhD thesis, Univ. Eindhoven, 1999.

[16] D. Kozen. Results on propositional μ-calculus. *Theoretical Computer Science.*, 27(3):333–354, 1983.

[17] R. A. Schmidt and D. Tishkovsky. Using tableau to decide expressive description logics with role negation. In *ISWC'07 + ASWC'07*, vol. 4825 of *LNCS*, pp. 438–451. Springer, 2007.

[18] R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. *Logical Methods in Computer Science*, 7(2):1–32, 2011.

[19] R. A. Schmidt and D. Tishkovsky. Using tableau to decide description logics with full role negation and identity, 2011. Manuscript, available at http://www.mettel-prover.org/papers/ALBOid.pdf.

[20] S. Schwendimann. A new one-pass tableau calculus for PLTL. In *Tableaux'98*, vol. 1397 of *LNAI*, pp. 277–291. Springer, 1998.

[21] C. Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *CP 2005*, vol. 3709 of *LNCS*, pp. 827–831. Springer, 2005.

[22] A. P. Sistla and E. M. Clarke. Complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[23] D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. Mettel2: Towards a tableau prover generation platform. In *Proc. PAAR'12*, 2012.

[24] P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, (110-111):119-136, 1985.

Appendix
Proofs for Section IV

Lemma 3: All TLC tableau rules except the rules ($E_{\Diamond}$-test) and ($E_{\mathcal{U}}$-test) preserve satisfiability of a tableau node.

Proof Sketch: Consider the rule ($E_{\Diamond}$). Suppose $f^k(0) : E_{\Diamond}(f^l(0), \varphi)$, $k \geq 0$ is satisfiable, that is, $(\sigma, k) \models E_{\Diamond}(f^l(0), \varphi)$ and, consequently, $(\sigma, l) \models \Diamond\varphi$. Thus, $(\sigma, k') \models E_{\Diamond}(f^l(0), \varphi)$ for every $k' \geq 0$. In particular, $(\sigma, k+1) \models E_{\Diamond}(f^l(0), \varphi)$ independently from whether (σ, k) validates φ or not. The other rules are considered similarly. ■

Lemma 2: $(\sigma_{\mathcal{B}}, i) \models \varphi$ for every persistent labelled formula $f^i(0) : \varphi$ occurring in $\mathcal{B}$.

Proof: We prove the lemma by induction on the structure of φ.

$\varphi = p$**.** This case follows immediately from the definition of $\sigma_{\mathcal{B}}$.

$\varphi = \neg\psi$**.** We consider the following cases:

$\psi = p$**.** This case follows from the definition of $\sigma_{\mathcal{B}}$.

$\psi = \neg\xi$**.** Since $f^i(0) : \neg\neg\xi \in \mathcal{B}$, the ($\neg\neg$)-rule has been applied to $f^i(0) : \neg\neg\xi$ or to a formula which has been subsequently rewritten to $f^i(0) : \neg\neg\xi$. In both cases, $f^i(0) : \xi \in \mathcal{B}$ and, by induction hypothesis, $(\sigma_{\mathcal{B}}, i) \models \xi$. Consequently, $(\sigma_{\mathcal{B}}, i) \models \varphi$.

$\psi = \xi \vee \eta$**.** This case is considered similarly to the previous one.

$\psi = \bigcirc\xi$**.** Because $f^i(0) : \neg\bigcirc\xi \in \mathcal{B}$ we have $f^i(0) : \bigcirc\neg\xi \in \mathcal{B}$ by the ($\neg\bigcirc$)-rule and $f^{i+1}(0) : \neg\xi$ appeared at some node of $\mathcal{B}$ by the ($\bigcirc$)-rule. Suppose $f^{i+1}(0)$ subsequently has been rewritten to a label $f^{k_i}(0)$ with k_i in $\{0, \ldots, n-1\}$ which persists in $\mathcal{B}$. Because $f^i(0) : \varphi$ is persistent in $\mathcal{B}$ this means that $i+1 = n$ and $k_i = k$. By the induction hypothesis, $(\sigma_{\mathcal{B}}, k) = (\sigma_{\mathcal{B}}, i+1) \models \neg\xi$. Therefore, $(\sigma_{\mathcal{B}}, i) \models \varphi$.

$\psi = \Diamond\xi$**.** Because $f^i(0) : \neg\Diamond\xi \in \mathcal{B}$ we have $f^i(0) : \Box\neg\xi \in \mathcal{B}$ by the ($\neg\Diamond$)-rule. By the ($\Box$)-rule $f^i(0) : \neg\xi \in \mathcal{B}$ and $f^{i+1}(0) : \Box\neg\xi$ appeared in a node of $\mathcal{B}$. By the induction hypothesis, $(\sigma_{\mathcal{B}}, i) \not\models \xi$. If $f^{i+1}(0)$ is persistent in $\mathcal{B}$ then again by the ($\Box$) $f^{i+1}(0) : \neg\xi \in \mathcal{B}$ and by the induction hypothesis $(\sigma_{\mathcal{B}}, i) \not\models \xi$. If otherwise $f^{i+1}(0)$ is subsequently rewritten to $f^{k_i}(0)$ in the branch $\mathcal{B}$ then $i+1 = n$ and $k_i = k$ as in the case for $\psi = \bigcirc\xi$. Therefore, $f^k(0) : \neg\xi \in \mathcal{B}$ by the ($\Box$)-rule and by the induction hypothesis $(\sigma_{\mathcal{B}}, k) = (\sigma_{\mathcal{B}}, i+1) \models \neg\xi$. Thus, continuing for all $j \geq i$, we obtain that $(\sigma_{\mathcal{B}}, j) \models \neg\xi$ for any $j \geq i$. That is, $(\sigma_{\mathcal{B}}, i) \models \neg\Diamond\xi$.

$\psi = \Box\xi$**.** Since $f^i(0) : \neg\Box\xi \in \mathcal{B}$ we also have that $f^i(0) : \Diamond\neg\xi \in \mathcal{B}$ by the ($\neg\Box$)-rule. Thus, there is $l \geq i$ such that $f^l(0) : E_{\Diamond}(f^l(0), \neg\xi)$ appeared in some node of $\mathcal{B}$ by the ($\Diamond$)-rule and either $i = l$ or $f^l(0)$ rewrites to $f^i(0)$ in $\mathcal{B}$. Because $\mathcal{B}$ is open there is a node of $\mathcal{B}$ where the ($E_{\Diamond}$) has been applied to $f^j(0) : E_{\Diamond}(f^s(0), \neg\xi)$ where $f^s(0)$ is a label rewritten from $f^l(0)$, and $f^j(0) : \neg\xi$ and $E_{\Diamond}(f^s(0), \neg\xi) \approx \Diamond\neg\xi$ are both appeared in $\mathcal{B}$. Because $f^i(0)$ is persistent in $\mathcal{B}$ we have that $f^s(0)$ rewrites to $f^i(0)$ in $\mathcal{B}$. Let t be a persistent label such that either $j = t$ or $f^j(0)$ is subsequently rewritten in $\mathcal{B}$ to $f^t(0)$. Thus, $f^t(0) : \neg\xi \in \mathcal{B}$ and $E_{\Diamond}(f^l(0), \neg\xi)$ is rewritten to $\Diamond\neg\xi$. By the induction hypothesis, $(\sigma, t) \not\models \xi$. We consider three cases.

$i \leq t$. In this case we immediately obtain $(\sigma, i) \models \neg\Box\xi$.

$i > t$ and $j = t$. We have $s > j$ in this case. Because $E_{\Diamond}(f^s(0), \neg\xi)$ is only propagated to labels which are the successors of $f^s(0)$, $f^j(0) : E_{\Diamond}(f^s(0), \neg\xi)$ with $j < s$ can appear in a tableau branch only if $f^j(0)$ is rewritten from some other label. Therefore $k \leq j < n$. Consider $t' \overset{\mathsf{def}}{=} (n-k) + j$. Clearly, $i \leq t'$. By the definition of σ we have $(\sigma, t') \not\models \xi$ and, consequently, $(\sigma, i) \models \neg\Box\xi$.

$i > t$ and $f^j(0)$ rewrites to $f^t(0)$. Thus, $k \leq t < n$ and $j = a \cdot (n-k) + t$ for some $a \geq 0$. Therefore, $(\sigma, j) \not\models \xi$. Clearly, $i < j$ and, hence, $(\sigma, i) \models \neg\Box\xi$.

$\psi = \xi\,\mathcal{U}\,\eta$**.** This case considered similarly to the case $\psi = \Diamond\xi$.

$\varphi = \bigcirc\psi$**.** This case is similar to the case $\varphi = \neg\bigcirc\xi$.

$\varphi = \Diamond\psi$**.** This case is considered similarly to the case $\varphi = \neg\Box\xi$.

$\varphi = \Box\psi$**.** Similarly to $\varphi = \neg\Diamond\xi$.

$\varphi = \xi\,\mathcal{U}\,\eta$**.** The case is similar to $\varphi = \neg\Box\xi$ with use of rules ($\mathcal{U}$), ($E_{\mathcal{U}}$), and ($E_{\mathcal{U}}$-test).

$\varphi = C^{\propto m}$**.** If C is not empty, that is, there $p \in C$ or $\neg p \in C$ for some proposition p. Assume that $p \in C$. By the (cut$^+$)-rule, either $f^i(0) : p \in \mathcal{B}$, or $f^i(0) : \neg p \in \mathcal{B}$. In the first case we have $f^i(0) : (C \setminus \{p\})^{\propto(m-1)} \in \mathcal{B}$. In the second case: $f^i(0) : (C \setminus \{p\})^{\propto m} \in \mathcal{B}$. Since length of $(C \setminus \{p\})$ is strictly smaller than the length of C we can apply the induction hypothesis and obtain that either $(\sigma_{\mathcal{B}}, i) \models (C \setminus \{p\})^{\propto(m-1)}$ and $(\sigma_{\mathcal{B}}, i) \models p$, or $(\sigma_{\mathcal{B}}, i) \models (C \setminus \{p\})^{\propto m}$ and $(\sigma_{\mathcal{B}}, i) \not\models p$. In both cases, it is clear that $(\sigma_{\mathcal{B}}, i) \models C^{\propto m}$.

Let now C be empty. Then $\#C = 0$. If $m = 0$ then $(\sigma_{\mathcal{B}}, i) \models C^{\propto m}$. Suppose that $m > 0$. If $\propto$ is the equality $=$ then $\mathcal{B}$ must be closed by the (cap)-rule that is a contradiction. If $\propto = \leq$ then clearly $(\sigma_{\mathcal{B}}, i) \models C^{\propto m}$. The case $m < 0$ is impossible because the constraint $\{\}^{\propto m}$ with negative m can be obtained only by sequential applications of the rules (C_+^+) and (C_+^-) from a constraint with non-negative restriction. However, in this case, a constraint with the restriction $\propto -1$ would appear and the closure rule (empty) would be applicable.

■