Tabled Resolution + Constraints: A Recipe for Model Checking Real-Time Systems*

Xiaoqun Du C.R. Ramakrishnan Scott A. Smolka
Department of Computer Science, SUNY at Stony Brook
Stony Brook, NY 11794–4400, USA
{vicdu, cram, sas}@cs.sunysb.edu

Abstract

We present a computational framework based on tabled resolution and constraint processing for verifying real-time systems. We also discuss the implementation of this framework in the context of the XMC/RT verification tool. For systems specified using timed automata, XMC/RT offers backward and forward reachability analysis, as well as timed modal mu-calculus model checking. It can also handle timed infinite-state systems, such as those with unbounded message buffers, provided the set of reachable states is finite. We illustrate this capability on a real-time version of the leader election protocol. Finally, XMC/RT can function as a model checker for untimed systems. Despite this versatility, preliminary benchmarking experiments indicate that XMC/RT's performance remains competitive with that of other real-time verification tools.

1. Introduction

In a recent paper [26], we showed that logic programming with tabulation can be used to construct an efficient model checker for untimed systems. In particular, we presented XMC, a model checker supporting XL (an extension of Milner's value-passing CCS) as the system specification language, and the alternation-free fragment of the modal mu-calculus as the property specification language.

XMC is written in XSB Prolog, where XSB [31] is a logic programming system developed at SUNY Stony Brook that extends Prolog-style SLD resolution with *tabled resolution*. The principal merits of this extension are that XSB terminates more often than Prolog (e.g. for all datalog programs), avoids redundant subcomputations, and computes the well-founded model of normal logic programs.

XMC is written in a highly declarative fashion. Essentially, it consists of two predicates `trans`, encoding the transitional semantics of XL terms, and `models`, defining when an XL term satisfies a given modal mu-calculus formula. All told, the model checker is written in less than 200 lines of XSB Prolog code.

Despite the high-level nature of XMC's implementation, we were able to—through the judicious use of source-level code optimizations—attain performance comparable to that of highly optimized model checkers such as Spin [20] and Murφ [9] on several examples, including some from the test suite contained in the standard Spin distribution. A subsequent paper [12] showed that XMC's performance can be improved even further by *compiling* XL specifications into a representation of the low-level automata. Another paper [24] showed how XMC could be extended to handle the full modal mu-calculus, i.e. with alternating fixed points of arbitrary nesting depth.

Our experience with XMC raises the following question: Can tabled logic programming be brought to bear on the problem of verifying *real-time systems* and what additional technologies are required? Given that many reactive systems of practical interest are real-time in nature, producing a logic-based framework for the verification of real-time systems can be viewed as an important next-step in XMC's evolution.

In this paper, we present the theory and implementation of a verification framework for real-time systems using the logic-based approach. Systems are specified as collections of timed safety automata [2, 17], a widely used specification formalism for real-time systems. Properties are specified in a real-time extension of the modal mu-calculus introduced in [28] which, for ease of discussion, we refer to as the "timed modal mu-calculus." We consider here only the

*This research was supported in part by NSF grants EIA-9705998 and CCR-9876242.

alternation-free fragment of the timed modal mu-calculus. As discussed below, this sub-logic is a very expressive one.

The main tangible outcome of this investigation is the XMC/RT verification tool. Like XMC, XMC/RT is written declaratively in XSB Prolog but uses the POLINE polyhedra package—a generic constraint solver for linear constraints over the reals [15]—for constraint processing. The combination of XMC and POLINE effectively gives us a Constraint Logic Programming (CLP) system with tabulation.

Despite its use of tabled resolution, XMC/RT is not simply an extension of XMC. In particular, it must construct and manipulate *region graphs* [2] in order to analyze timed-automata-based specifications of real-time systems, a requirement not present in XMC. Moreover, convex sets of constraints are used to represent regions, and this is the reason that constraint solving is needed in the XMC/RT framework.

Region graphs represent a finite quotient of the inherently infinite state space underlying a timed automaton. Inspired by [28], XMC/RT constructs region graphs *locally*, or on-the-fly, yielding a quotient that is *as coarse as possible* in the following sense: refinements of the quotient are carried out only when necessary to satisfy *clock constraints* appearing in the logical formula or timed automaton used to represent the system under investigation. The local approach can result in the exploration of significantly fewer regions on reasonable examples when compared to a global algorithm, and is consistent with the use of a top-down resolution-based strategy for computing fixed points. This was also the case with XMC in the untimed case.

The advantages of the approach to verifying real-time systems embodied in XMC/RT lie mainly in its versatility and include the following:

- Different styles of analysis can be readily implemented in a single framework, including forward reachability, backward reachability, and model checking of timed modal mu-calculus formulas. Even in the alternation-free case, the expressiveness of the timed modal mu-calculus goes beyond reachability, allowing one to specify properties such as liveness and bounded liveness.
- Timed infinite-state systems, such as those with unbounded message buffers, can be analyzed within this framework if the set of reachable states is finite. This is because Prolog terms are used to represent system states and buffers are encoded as dynamic Prolog list data structures. We illustrate this point on a real-time version of leader election.
- Finite-state *untimed* systems can be analyzed with little overhead. In particular, given a specification that does not utilize clocks, XMC/RT will conduct the analysis in a fashion virtually identical to the resolution-based style deployed by XMC. This capability is illustrated on the i-protocol [11] and (untimed) leader election.

In contrast, most extant tools for verifying real-time systems focus exclusively on either reachability analysis or model checking, and cannot handle specifications that are either untimed or contain unbounded structures. Related work is discussed in greater detail below.

Somewhat surprisingly, XMC/RT's versatility is achieved without incurring a severe performance penalty. In particular, preliminary experimental data shows that XMC/RT is competitive performance-wise with HyTech and Uppaal on several standard benchmarks. For untimed systems, XMC/RT takes less than 50% additional time compared to XMC.

Related Work

Several researchers have also used constraint logic programming for the verification of real-time and other infinite-state systems. Pontelli and Gupta [14] model real-time systems as CLP programs, and safety properties are verified using reachability queries. Other kinds of correctness properties, such as liveness, are not considered. Their formulation exploits CLP's handling of constrained variables and hence can verify parametric systems (computing, for instance, values of parameters for which a given property holds). Since, however, their tool is based on a traditional Prolog system with constraints, no termination guarantees are given.

Delzanno and Podelski [8] encode discrete, infinite-state systems (e.g. the bakery algorithm) as CLP programs, and verify CTL properties by computing least and greatest fixed points of the logical consequence operator. The fixed-point computations are implemented as meta-programs in SICStus Prolog. While these computations are intrinsically global, Magic Set transformations [27] are used to make them more goal-directed. In contrast, we use XSB's evaluation strategy to directly compute fixed points in a goal-directed manner, but add constraint processing via meta-programming. This approach enables XMC/RT's performance to approach that of a finite-state model checker (XMC) when the system to be verified has no real-time components. It should be noted that timed systems can be expressed (via explicit manipulation of clocks) using the notation in [8]. Mukhopadhyay and Podelski [25] give

sufficient conditions under which reachability analysis terminates without constructing region graphs. However, the timed modal mu-calculus model checker in XMC/RT can verify a larger class of formulas (such as livelock-freedom) compared to the CTL model checker of [8] or the reachability checker of [25]. As discussed in Section 3, this difference is a significant one computationally.

Urbina [30] models hybrid systems as CLP programs. Various properties of hybrid systems can then be verified by top-down or bottom-up evaluation of the CLP programs. This approach is also based on CLP systems without tabling.

Bjorner et al. [4] and Kesten et al. [21] use deductive approaches to verify real-time systems. They model systems using Clocked Transition Systems, which are fair transition systems extended with clock variables, and use verification rules and verification diagrams to establish the validity of Linear Temporal Logic formulas.

Kwak et al. [22] show how process-algebraic methods can be used to reduce the schedulability problem for real-time systems to a set of equations, a solution to which yields the values of the parameters that make the system schedulable. Equations are solved using integer programming or constraint logic programming.

Another class of model checkers, including Uppaal [23], and the Concurrency Factory [28], use *difference bound matrices* (DBMs) to represent constraints. While a DBM is a well-tuned data structure for representing constraints that relate at most two variables, and hence ideal for real-time system verification, its use limits these tools to non-parametric analysis. In contrast, HyTech [16] and Kronos [33] are based on a general (polytope) representation for linear constraints over reals. As noted above, XMC/RT also uses a generic constraint solver for linear arithmetic constraints over reals, namely POLINE [15].

Organization We begin in Section 2 with a brief overview of timed safety automata, the timed modal mu-calculus, and tabled logic programming. We introduce the constraint-logic-based formulation of XMC/RT and its implementation in Section 3. Experimental results for real-time benchmarks, unbounded real-time systems, and untimed systems appear in Section 4. We offer some concluding remarks in Section 5.

A full version of the paper can be found in [13]. A prototype of XMC/RT is available from the authors upon request, and will be included in the XMC distribution at `www.cs.sunysb.edu/~lmc`

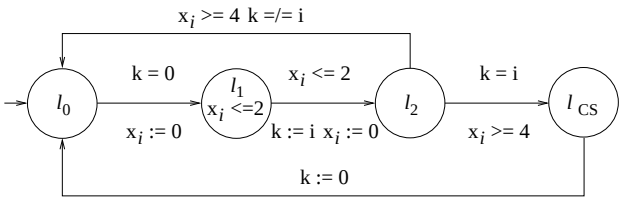

Figure 1. Timed Safety Automaton for one process of Fischer's mutual exclusion protocol.

2. Preliminaries

In this section, we recall the notion of timed safety automaton, give the syntax and semantics of the timed modal mu-calculus, briefly review tabled logic programming, and sketch the issues that arise in integrating constraints with tabled resolution.

2.1. Timed Safety Automata

A *Timed safety automaton* (TSA) is a finite-state automaton extended with real-valued clocks. States of the automaton are associated with identifiers called *locations*. Each transition has an *action* label drawn from a finite alphabet, and may also specify a subset of clocks that are reset upon taking the transition. The conditions under which a transition is enabled are given by *constraints* on clock values. A constraint is a (possibly empty) conjunction of base constraints of the form $x \odot c$ where x is a clock, c is an integer constant, and $\odot$ is taken from the set $\{\leq, \geq, <, >\}$. Each state of the automaton also has an associated *location invariant*: a constraint on clock values that must be satisfied for the automaton to remain in that state.

Intuitively, a TSA operates by taking transitions from location to location. Executing a transition takes no time. If no transitions are taken, time progresses by uniformly incrementing every clock value by an arbitrary real number.

For conciseness of expression, TSAs are usually extended with *discrete* variables, which are non-clock variables that take values over finite domains. This does not increase the expressive power of TSA, since discrete variables can be eliminated by expanding the automaton's state space.

The example TSA depicted in Figure 1 corresponds to one process (with process id i) in Fischer's mutual exclusion protocol [1]. The automaton has one clock x_i and one discrete variable k. Its start state has location l_0 and the

state labeled l_{CS} represents the process's critical section. The clock x_i is reset on transitions from l_0 to l_1 and l_1 to l_2. The transition l_1 to l_2 is enabled only when the value of clock x_i is less than or equal to 2. State l_1's location invariant decrees that the automaton cannot remain in l_1 once the clock value exceeds 2.

The semantics of a TSA is given in terms of a dense *labeled transition system* (LTS) whose states correspond to a location in the TSA along with a unique valuation of its clocks. The LTSs are dense since the number of states, as well as the number of transitions from any state may be uncountable. For example, the LTS induced by the TSA in Figure 1 has distinct states of the form $\langle l_1, v \rangle$ for each real number $v \leq 2$. It has transitions from $\langle l_1, v \rangle$ to $\langle l_1, v + \delta \rangle$ for all real values v and δ such that $v < v + \delta \leq 2$; $\langle l_1, v + \delta \rangle$ is called a *time successor* of $\langle l_1, v \rangle$. The LTS also has transitions from $\langle l_1, v \rangle$ to $\langle l_2, 0 \rangle$ for each $v \leq 2$ and $\langle l_2, v \rangle$ to $\langle l_{CS}, v \rangle$ for each $v \geq 4$. The destinations of these transitions are known as the *transition successors* of the respective sources.

2.2. Timed Modal Mu-Calculus

The timed modal mu-calculus, introduced in [28] and derived from the real-time logics of [17, 18], adds time modalities $\langle \epsilon \rangle$ and $[\epsilon]$ and resettable formula clocks to the modal mu-calculus. Our encoding of the alternation-free fragment of the timed modal mu-calculus uses the following syntax for formulas:

```
F  -->  tt | ff | atomic(C) | form(X) |
        and(F, F) | or(F, F) | neg(F) |
        diam(Act,F) | box(Act,F) |
        epsdiam(F) | epsbox(F) | reset(Z,F)
```

`X` is a logical variable and `tt` and `ff` are propositional constants; `atomic(C)` is a base formula where `C` is either an atomic proposition or a constraint over system and formula clocks; `and`, `or`, and `neg` are standard logical connectives; `diam(Act,F)` (formula `F` holds in a state reached after action `Act` is taken *without any delay, i.e. passage of time*) and `box(Act,F)` (formula `F` holds in all states reached after action `Act` *without delay*) are dual discrete modal operators; `epsdiam(F)` (after some delay formula `F` holds) and `epsbox(F)` (after every delay formula `F` holds) are dual real-time modal operators; `reset(Z,F)` defines a new clock local to formula `F`.

Logical variables are provided for definitions using fixed-point equations of the form `X += F` (least fixed point) and or `X -= F` (greatest fixed point). If the defining equations are stratified with respect to greatest and least fixed point operators, the equation system is called non-alternating.

Given a TSA T, the semantics of a formula ϕ is given with respect to the dense LTS induced by the new TSA T' constructed from T by adding the clocks defined in ϕ. As with untimed modal logics, the semantics of a timed modal mu-calculus formula ϕ is given by the set of states in the LTS that model ϕ, and this state set is defined inductively on the structure of ϕ. We can derive a computational procedure for evaluating the semantics of ϕ by choosing an appropriate (constraint) representation of the states sets.

2.3. Tabled Logic Programming with Constraints

Tabled-resolution methods in logic programming [5, 29, 6] address the well-known shortcomings of the SLD evaluation mechanism of Prolog, namely, susceptibility to infinite looping, redundant subcomputations, and inadequate semantics for negation. When tabled resolution is used in XSB (by declaring particular predicates to be tabled), the system automatically maintains a table of predicate invocations and answers, using the table for all equivalent invocations after the first one. Many programs that would loop infinitely in Prolog will terminate in XSB because XSB calls a tabled predicate with the same arguments only once, whereas Prolog may call such a predicate infinitely often. XSB computes the well-founded model for programs with negation: we exploit this ability to compute greatest fixed points as the negations of least fixed points.

The XSB system has no in-built support for constraint processing. Hence, we integrate an external constraint solver with the native logic programming engine. Our integration separates constraint processing from the control aspects of fixed-point evaluation. The constraint solver determines an internal representation for constraints, and assigns *handles* to constraints. The XSB engine identifies constraints only by these handles, and invokes the solver for any operation that involves constraint handles.

3. CLP-Based Formulation and Implementation

In this section, we describe the CLP-based formulation of the verification framework for real-time systems that forms the basis for XMC/RT.

Reachability: Consider the XMC system for verifying (untimed) properties of finite-state systems. There we encoded reachability analysis and the semantic equations for

the modal mu-calculus in Horn-clause notation, and evaluated the logic program using tabled resolution. The clauses from the XMC model checker [26] for reachability analysis are as follows:

```
:- table reach/2.
reach(X,Y) :- trans(X,_,Y).
reach(X,Y) :- reach(X,Z), trans(Z,_,Y).
```

`reach/2` indicates that `reach` is a binary predicate. `trans(X,_,Y)` means that there is a transition from `X` to `Y` by any label.

`diam` formulas can be handled similarly to the case of reachability analysis. The semantic equation for a `diam` formula is:

$$[\![\text{diam}(\alpha, F)]\!]e = \{s \mid \exists t \ s \xrightarrow{\alpha} t \text{ and } t \in [\![F]\!]e\}$$

The corresponding clause from XMC is:

```
:- table models/2.
models(S, diam(Act,F)) :- trans(S, Act, T),
                          models(T, F).
```

specifying that a state `S` models a formula `diam(Act,F)` if there exists a state `T` such that there is transition from `S` to `T` with label `Act` and `T` models `F`.

This encoding can be used as is for timed systems if one interprets the predicates `reach` and `models` as constraint logic programs; i.e. we interpret `S` not as a single state, but as a (finite) representation of a (possibly infinite) set of states. For example, consider the (infinite) LTS induced by the TSA in Figure 1. A finite representation such as $\langle l_2, x_i \geq 4\rangle$ represents the infinite set of LTS states with location l_2 and with value of x_i no less than 4. This encoding can be evaluated by using any complete inference procedure for logic programs, such as bottom-up evaluation or tabled resolution, as long as the `trans` relation produces only finitely many distinct constraints `T` for any given constraint `S`.

We use this encoding for verifying reachability and safety properties in the timed case. This approach is similar to the approaches taken in [14, 8] for safety properties and CTL formulas, respectively. Backward reachability analysis for real-time systems can also be encoded in a similar fashion, with the help of a predicate that computes transitions between constraints backwards, i.e. given a constraint representing a set of target states, it computes a constraint representing the set of source states.

The formulation of forward and backward reachability as well as `diam` formulas for real-time systems has been implemented in XSB using an external library for constraint solving. Specifically, we used POLINE, a library for manipulating convex polyhedra over n-dimensional real space [15] for constraint solving, and connected to it using XSB's foreign-language interface.

Timed Modal Mu-Calculus: Reachability analysis and `diam` formulas involve only existential quantifications. We now consider universal quantifications. The `box` formulas in untimed modal mu-calculus contain universal quantifications over finite number of target states by transitions labeled by `Act`. These formulas can be handled using the Prolog construct `findall`, which is used to collect all target states of a given state. However, the semantics of the `epsbox` modality is defined as:

$$[\![\texttt{epsbox(}F\texttt{)}]\!]e = \{\langle l,\pi\rangle \mid \forall \delta \geq 0 \ \langle l,\pi\rangle \xrightarrow{\epsilon} \langle l,\pi+\delta\rangle \Rightarrow \langle l,\pi+\delta\rangle \in [\![F]\!]e\}$$

Since the universal quantification is over (uncountable) clock values, the operational encoding using `findall` is not applicable.

We overcome this problem by encoding the elimination of the universally quantified variable directly as a built-in constraint operation. Let `S` be a set of states with the common location l and *inv*(`S`) be the location invariant of l. Let v, v' represent states and `D` be a variable such that `D` $\notin C$, where C is the set of clock variables. Furthermore, let v + `D` represent a state where the value of all clock variables in v is incremented by `D`. We define `univ_elim(D, S, Goal, SD, SS)` such that, given `SD`, a set of states that is the set of all solutions to a Prolog predicate `Goal`, `SS` is the set of states $\{v \mid v \in$ `S` $\wedge \ \forall$`D` $\in \mathbf{R}(v' = v +$ `D` $\wedge\ v' \in$ *inv*(`S`)$) \Rightarrow (v' \in$ `SD`$)\}$.

The `univ_elim` operation can be implemented through two difference operations over polyhedra and existential elimination. Notice that the `univ_elim` operator requires direct manipulation of the solutions to `Goal`. Typically `Goal` is a call to the predicate `models`. Thus instead of using a binary predicate `models` and let the CLP system implicitly return answers by binding additional constraints, we modify the binary `models` predicate to derive a ternary predicate `models(S, F, SS)`. Given a set of states `S`, and a formula `F`, the ternary `models` *explicitly* returns `SS`, which contains only states in `SS` that models `F`. in order for the correct computation of the elimination of universal quantification, `models(S, F, SS)` must also return all states in `S` that model `F` all at once. We ensure this by defining `models` as an aggregation over another relation `models1`, which is identical to `models` except that it is not required to return all states in `SS` that models `F` at once. The most interesting fragments of the definition of `models` and `models1` are given below.

```
:- table models/3.
models(S, F, SS) :-
  union(T, models1(S, F, T), SS).

% formula recursion with lfp
models1(S, form(F), SS) :-
  F += E, models(S, E, SS).

% negation (due to greatest fixed points)
models1(S, neg(F), SS) :-
  models(S, F, NegSS), diff(S, NegSS, SS).

models1(S, or(F1, F2), SS) :-
  models(S, F1, SS) ; models(S, F2, SS).

% universal transition modality
models1(S, box(Act, F), SS) :-
  split(S, Act, LS),
  member(S1, LS),
  findall(TS, trans(S1, Act, T), TS),
  all_models(TS, F, S1, Act, SS).

all_models([], _, _, _, []).
all_models([T0|Rest], F, S, Act, SS) :-
  models(T0, F, TS0),
  inverse_trans(TS0, Act, S, SS1),
  all_models(Rest, F, S, Act, SS2),
  conjunction(SS1, SS2, SS).

% universal time modality
models1(S, epsbox(F), SS) :-
  univ_elim(D, S,
            (trans(S, e(D), T),
             models(T, F, TS),
            ), TS, SS).
```

In the above definitions, `union(V, G, R)` is a relation such that, given a goal `G` that contains a variable `V`, `R` is the canonical representation of all states s such that `G`$[s/V]$ is true; `diff` computes the difference between two constraints; `inverse_trans(TS, Act, S, SS)` is such that `SS` represents all states s in `S` such that $s \xrightarrow{\texttt{Act}} t$ and t is a state represented by `TS`; `split(S, Act, LS)` is such that, given a constraint `S`, `LS` is a finite list $[c_1, c_2, \ldots, c_n]$ that represents a partition of `S` such that $\forall 1 \leq i \leq n$ (a) $c_i \Rightarrow$ `S`, and (b) all states represented by c_i are identical w.r.t. `Act`; `conjunction(SS1, SS2, SS)` is such that `SS` is the conjunction of constraints `SS1` and `SS2`. `all_models(TS, F, S, Act, SS)` is such that, `SS` is a subset of `S` such that the targets of `SS` by `Act`-transitions, which are in the list `TS`, models `F`.

Correctness of the model-checking algorithm The key to the correctness proof is the definition of a semantic function h lifting the point-based semantics of the timed modal mu-calculus to a region-based semantics. Intuitively, given a formula Φ and a set of states S, h returns the subset of states in S that models Φ. The Horn clauses appearing in the formulation of the model checker can be seen as a direct translation of the semantic function h. See [13] for the details.

Optimizations The logic program implementing the model checker can be subjected to several optimizations, two of which are described below.

Call abstraction: Logically speaking, a query of the form $p(t)$ is equivalent to the query $p(X)$, $X = t$, for any term t. In a tabled logic programming environment, such an abstraction can improve performance since two distinct queries of the form $p(t_1)$ and $p(t_2)$ with overlapping answer sets now share the answer computations. In our model checker, a call to `models(S,F,SS)` can be replaced by `models(S',F,SS')` where `S'` is the set of states obtained by abstracting from `S` all clock constraints except the location invariants.

Early completion: The aggregation predicate `union` collects answers from *all* the rules that match the current call to `models1`. Clearly, answer generation can be stopped as soon as the partial union covers the solution space. For instance, consider a call of the form `models1(S, or(X1, X2), SS)`, which results in a call to `models(S, X1, SS)`. If this call results in an answer `SS` such that `S` $\Rightarrow$ `SS`, then the subsequent call to `models(S, X2, SS)` can be eliminated.

4. Experimental Results

In order to gauge XMC/RT's performance as a real-time verification tool, we consider two example systems that have become widely accepted benchmarks in the real-time community: (1) Fischer's mutual exclusion protocol [1] for 2, 3, and 4 processes; and (2) a bridge-crossing system, adapted from [32]. Specifically, we compare XMC/RT's performance on these benchmarks to that of HyTech version 1.04, and UPPAAL version 3.0.40. These are the latest versions of HyTech and UPPAAL currently available. Both tools perform reachability analysis: forward and backward reachability in the case of HyTech; forward reachability only in the case of UPPAAL.

Fischer's mutual exclusion protocol Fischer's mutual exclusion protocol controls access to a critical section using a turn variable k and by imposing timing constraints on when the turn variable can be modified. The protocol is modeled as the parallel composition of n TSAs, each of which has the structure of the TSA depicted in Figure 1. We verify three properties of the protocol.

- *Safety*: at most one process is in the critical section at any time.
- *Possibility*: from all states s reachable from the initial state, a state with exactly one process in the critical section can eventually be reached.
- *Liveness*: from all states s reachable from the initial state, every evolution of the system eventually reaches a state with exactly one process in the critical section.

The bridge-crossing system In the bridge-crossing system, a controller schedules the crossing of two trains across a bridge that has only one track. The system is modeled as the parallel composition of three TSAs: a controller process and two train processes. The controller uses synchronization signals to control the movement of the trains, and it keeps a list L of trains on the bridge or waiting to cross the bridge. Initially, both trains are far away, the bridge is free, and L is empty. Train i emits the signal $appr_i$ as it nears the bridge. If the bridge is free, the controller allows the train to proceed. Otherwise, the controller sends the signal $stop_i$ to stop the train. It will send the signal go_i to let the train cross the bridge when the the bridge eventually becomes free.

We consider three correctness properties.

- *Safety*: the trains are not on the bridge at the same time.
- *Liveness*: a train will always eventually cross the bridge.
- *Bounded Liveness*: a train leaves the crossing within 50 time units of its approach.

Benchmarking results Table 1 contains the running times we obtained for Fischer's mutual exclusion protocol and the bridge-crossing system. In the case of Fischer's protocol, we considered 2, 3 and 4 processes. All properties of interest can be expressed in the timed modal mu-calculus and can thus be verified using XMC/RT's model checker for that logic. Safety can also be verified using forward or backward reachability, while the possibility property can be verified by combining forward and backward reachability. Because UPPAAL does not support backward reachability, possibility results are not given for that tool. By introducing a monitor automaton into the bridge-crossing system, the bounded-liveness property can also be verified using forward or backward reachability.

Liveness cannot be verified using reachability analysis and thus results in this case are presented only for XMC/RT. Because for both HyTech and XMC/RT, backward reachability is much faster than forward reachability, we present the data obtained from backward reachability whenever a property can be verified using either technique.

All data was obtained on a Sun Enterprise 4000 with 2GB memory running Solaris 5.2.6. Two sets of data were obtained for UPPAAL: without the -D option (UPPAAL1) and with the -D option (UPPAAL2). The -D option causes UPPAAL to disable deadlock warnings, and results in markedly lower execution times in some of the benchmarking runs. Overall, the data indicates that XMC/RT is competitive with HyTech on the Fischer and bridge-crossing benchmarks but significantly less so with UPPAAL, with the exception of the 4-process Fischer's protocol when the -D option is not used. This is to be expected in the following sense. Both XMC/RT and HyTech rely on POLINE as a general constraint solver for convex polyhedra. UPPAAL, on the other hand, uses clock difference diagrams (CDD), a BDD-like data structure for representing and efficiently manipulating certain non-convex subsets of the Euclidean space, such as those encountered during verification of timed automata. The use of a general (polytope) representation for linear constraints over the reals permits *parametric analysis* (available in HyTech planned for XMC/RT) to be performed, while more restricted data structures such as DBMs (difference bound matrices) and CDDs do not. DBMs and CDDs, however, are more efficient.

In further comparing XMC/RT and HyTech, both tools exhibit similar performance on backward reachability. Comparable performance is also obtained in these cases using XMC/RT's model checker and this is likely attributable to the call-abstraction optimization discussed above. The table also shows that XMC/RT is slower than HyTech for verifying the the possibility property of the 4-process Fischer's protocol. This is due to an inefficient subsumption check in our prototype implementation. In particular, every time a region is visited, a table lookup is performed to ascertain whether the region has been visited previously. Indexing techniques that can speed up this check are currently under investigation. We plan to implement them as a part of a fully integrated tabled logic-programming system with constraints.

The verification of bounded liveness using XMC/RT's timed mu-calculus model checker is slower than the corre-

System	Property	XMC/RT		HyTech	UPPAAL1	UPPAAL2
		mu-calc	reach		w/o -D	with -D
Fischer 2 proc	Safety	0.08	0.05	0.11	0.01	0.01
	Possibility	0.24	0.20	0.44	-	-
	Liveness	0.36	-	-	-	-
Fischer 3 proc	Safety	0.84	0.46	0.99	0.30	0.08
	Possibility	4.13	4.08	5.30	-	-
	Liveness	1.54	-	-	-	-
Fischer 4 proc	Safety	11.4	9.0	8.7	21.6	2.24
	Possibility	105	201	103	-	-
	Liveness	33.4	-	-	-	-
Bridge Crossing	Safety	0.19	0.17	0.14	0.03	0.02
	Bounded Liveness	7.0	1.19	1.19	2.18	0.04
	Liveness	0.8	-	-	-	-

Table 1. Running times (seconds) for the verification of Fischer's mutual exclusion protocol and the bridge-crossing system. "reach" stands for reachability.

sponding (unbounded) liveness analysis. This is because the time bound in the formula induces extra splitting of regions and also reduces the potential for the early-completion optimization.

4.1. A Real-Time Version of Leader Election with Unbounded Message Queues

In a leader-election protocol [10], n nodes, connected by *unbounded* buffers, form a ring. The nodes are identical except they have unique identifiers. Each node sends messages to its neighbor on the right and receives messages from its neighbor on the left, comparing the messages with its local variables, until eventually exactly one node is elected as the leader. Although the protocol uses unbounded buffers as the communication media, each buffer in fact holds only a finite number of messages during the protocol's execution.

We extend the leader-election protocol by placing timing constraints on the sending actions of the nodes so that each message must be transmitted within 5 time units. This real-time protocol has the property that there is an upper bound on the elapsed time before a leader will be elected.

In our specification of the protocol, there is one process per node and one process per medium; each process is modeled as a TSA. Once a buffer becomes non-empty, the corresponding TSA is required to send a message within 5 time units. The unbounded buffers of the medium processes are modeled naturally in XMC/RT using Prolog lists. To illustrate this technique, the following clause of the `trans` predicate describes how the local variable `Buf` of a medium process is updated when message `Msg` is received over channel `chan`. The process moves from location `empty` to `nonempty` upon this transition.

```
trans((empty(Buf),R), in(chan(Msg)),
        (nonempty([Msg|Buf]),R)).
```

We used XMC/RT's model checker for the timed modal mu-calculus to verify both unbounded and bounded liveness for a two-node leader-election system. The unbounded liveness property requires that a leader is eventually elected, while the bounded liveness property requires that a leader is elected within 40 time units. The time bound is 40 because the protocol requires two rounds of message passing before a leader can be elected. In each round, the nodes simultaneously transmit two messages each; this, plus the time required by the medium processes, leads to the requirement of 20 time units per round.

XMC/RT needed 1.1 seconds for the verification of the unbounded liveness property and 13.4 seconds for the bounded liveness property. As discussed above, although there are unbounded buffers in the protocol, XMC/RT was able to verify these properties because it combines local model checking with the dynamic data structures (lists) offered by the XSB tabled logic programming system.

4.2. XMC/RT for the Verification of Untimed Systems

Recall (Section 3) that in XMC/RT, each predicate invocation of the form `models(SS,F,SR)` computes a maximal subset of states in `SS` that models formula `F`. More precisely, `SS` is a set of states (from the dense LTS representing

System	Version	Formula	XMC	XMC/RT
Leader Election	7 processes	one_leader	14.2	18.3
		af_leader	14.1	15.4
		ef_leader	11.4	14.1
i-Protocol	W=1, buggy	livelock	0.05	0.07
	W=1, fixed	livelock	12.1	17.6
	W=2, buggy	livelock	0.32	0.41
	W=2, fixed	livelock	201.7	305.8

Table 2. Running times (seconds) for the verification of untimed systems.

the semantics of the TSA under investigation) corresponding to a pair $\langle l, R\rangle$, where l is a location in the TSA and R is a region (constraint). In contrast, in XMC, the corresponding invocation is of the form `models(S,F)` which simply succeeds or fails depending on whether the single state represented by `S` models `F`. Now, when `models(SS,F,SR)` is invoked and the Prolog variable representing the region underlying `SS` is *unbound*, this region will be interpreted as the constraint "true". In this case, the rules in XMC/RT for the untimed operators of the timed modal mu-calculus do not perform any computations on regions, and the behavior of XMC/RT thus closely resembles that of XMC. That is, XMC/RT can effectively analyze *untimed* systems by ensuring that region variables are unbound. In order to estimate the overhead attributable to this more complex formulation of the `models` predicate as well as to XMC/RT's constraints interface, we compared the performance of XMC/RT with that of XMC for verifying untimed systems.

Table 2 documents the performance of XMC and XMC/RT on the 7-process leader election protocol (adapted from the Spin benchmark suite), and on the sliding-window protocol found in GNU uucp called the i-protocol. Three basic properties of leader election were verified. For the i-protocol, we checked for the existence of a livelock error, for "fixed" (correct) and "buggy" versions of the protocol, window sizes 1 and 2. The data for XMC in Table 2 is taken from [12].

Our results indicate that the performance of the real-time model checker XMC/RT approaches that of the finite-state model checker XMC. It should be noted that XMC's performance for the i-protocol is comparable to that of Spin and Murφ [11, 12, 19]. Thus, we see that in XMC/RT, the structures for handling real-time verification lead to very little overhead when applied to finite-state systems.

5. Conclusions

We have shown how a careful integration of tabled resolution and constraint processing can yield a computational framework that is well suited to model checking real-time systems. Our real-time model checker, XMC/RT, is written in a highly declarative manner (approx. 350 lines of Prolog code, not counting, of course, the POLINE package), utilizes a local approach to region-graph construction, and can handle real-time and untimed systems with unbounded structures.

Future work includes integrating constraints directly into the XSB tabled logic programming system, i.e. *at the engine level* [7]. We expect considerable improvement in performance in an XMC/RT built on top of such a CLP system, and would like to verify this conjecture experimentally.

References

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real-time. In *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*, Berlin, 1991. Springer-Verlag.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] R. Alur and T. A. Henzinger, editors. *Computer Aided Verification (CAV '96)*, volume 1102 of *Lecture Notes in Computer Science*, New Brunswick, New Jersey, July 1996. Springer-Verlag.

[4] N. Bjørner, Z. Manna, H. Sipma, and T. Uribe. Deductive verification of real-time systems using STeP. In *Proc. 4th Intl. AMAST Workshop on Real-Time Systems, ARTS-97*, volume 1231 of *Lecture Notes in Computer Science*, pages 22–43, May 1997.

[5] R. Bol and L. Degerstadt. Tabulated resolution for well-founded semantics. In *Proceedings of the Symposium on Logic Programming*, 1993.

[6] W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, Jan. 1996.

[7] B. Cui and D. S. Warren. Mutable terms in a tabled logic programming system. In *International Conference on Logic Programming*, Las Cruces, New Mexico, USA, Nov. 1999.

[8] G. Delzanno and A. Podelski. Model checking in CLP. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99), Lecture Notes in Computer Science*, volume 1579, pages 223–239, Amsterdam, Mar. 1999.

[9] D. L. Dill. The Murφ verification system. In Alur and Henzinger [3], pages 390–393.

[10] D. Dolev, M. Klawe, and M. Rodeh. An $o(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *Journal of Algorithms*, 3:245–260, 1982.

[11] Y. Dong, X. Du, Y. Ramakrishna, C. Ramakrishnan, I. Ramakrishnan, S. A. Smolka, O. Sokolsky, E. W. Stark, and D. S. Warren. Fighting Livelock in the i-Protocol: A Comparative Study of Verification Tools. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99), Lecture Notes in Computer Science*, volume 1579, pages 74–88, Amsterdam, Mar. 1999.

[12] Y. Dong and C. Ramakrishnan. An optimizing compiler for efficient model checking. In *Formal Description Techniques For Distributed Systems and Communication Protocols & Protocol Specification, Testing, And Verification (FORTE/PSTV'99)*, 1999.

[13] X. Du. *Tabled Resolution and Constraints for Model Checking Real-Time Systems and Infinite-State Systems*. PhD thesis, State University of New York at Stony Brook, Aug. 2000. Available at `www.cs.sunysb.edu/~vicdu/thesis.ps.gz`.

[14] G. Gupta and E. Pontelli. A Constraint Based Approach for Specification and Verification of Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1997.

[15] N. Halbwachs, Y. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

[16] T. A. Henzinger, P. H. Ho, and H. Wong-Toi. "HyTech: A model checker for hybrid systems". *International Journal on Software Tools for Technology Transfer*, 1(2):110–122, October 1997.

[17] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[18] U. Holmer, K. G. Larsen, and Y. Wang. Deciding properties of regular real timed processes. In *Proceedings of CAV'91*. LNCS 575, 1991.

[19] G. J. Holzmann. The engineering of a model checker: the Gnu i-Protocol case study revisited. In *Proc. of the 6th Spin Workshop*, volume 1680 of *Lecture Notes in Computer Science*, Toulouse, Sept. 1999.

[20] G. J. Holzmann and D. Peled. The state of SPIN. In Alur and Henzinger [3], pages 385–389.

[21] Y. Kesten, Z. Manna, and A. Pnueli. Verifying Clocked Transition Systems, Hybrid Systems III. In *Proceedings of the Eight International Conference on Computer Aided Verification (CAV '96)*, Vol. 1102 of *Lecture Notes in Computer Science*, volume 1066 of *Lecture Notes in Computer Science*, pages 13–40, 1996.

[22] H. H. Kwak, J. Y. Choi, I. Lee, A. Philippou, and O. Sokolsky. Symbolic schedulability analysis of real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 409–418, December 1998.

[23] K. G. Larsen, P. Pettersson, and W. Yi. Model checking for real-time systems. In *Proc. of Fundamentals of Computation Theory*, number 965 in Lecture Notes in Computer Science, pages 62–88, Aug. 1995.

[24] X. Liu, C. R. Ramakrishnan, and S. A. Smolka. Fully local and efficient evaluation of alternating fixed points. In *Proceedings of the Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '98), Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[25] S. Mukhopadhyay and A. Podelski. Beyond region graphs: Symbolic forward analysis for timed automata. In C. Pandurangan, editor, *Proceedings of Foundations of Software Technology and T heoretical Computer Science (FST & TCS '99)*, Chennai, India, December 1999.

[26] Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. W. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In *Proceedings of the 9th International Conference on Computer-Aided Verification (CAV '97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 143–154, Haifa, Israel, July 1997. Springer-Verlag.

[27] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. Efficient bottom-up evaluation of logic programs. In P. D. Wilde and J. Vandewalle, editors, *Computer Systems and Software Engineering: State-of-the-Art*. Kluwer Academic, 1992.

[28] O. Sokolsky and S. A. Smolka. Local model checking for real-time systems. In *Proceedings of the 7th International Conference on Computer-Aided Verification*, volume 939 of *Lecture Notes in Computer Science*. American Mathematical Society, 1995.

[29] H. Tamaki and T. Sato. OLDT resolution with tabulation. In *International Conference on Logic Programming*, pages 84–98. MIT Press, 1986.

[30] L. Urbina. Analysis of hybrid systems in CLP(R). In *Proceedings of the 2nd nternational conference on principles and practice of constraint programming (CP96)*, volume 1118 of *Lecture Notes in Computer Science*, pages 451–467, 1996.

[31] XSB. The XSB logic programming system. Available at `www.cs.sunysb.edu/~sbprolog`.

[32] W. Yi, P. Pettersson, and M. Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In D. Hogrefe and S. Leue, editors, *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*, pages 223–238. North–Holland, 1994.

[33] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(2):123–133, October 1997.