CS780 Project: The Weighted Automata Network Analyzer

Peter Kemper
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187, USA
kemper@cs.wm.edu

Abstract

Weighted automata generalize a number of concepts found in discrete event dynamics systems of various kind. Semirings are used as a base algebra for describing weights which implies a variety different interpretations for particular application cases. In this project, we want to explore how theoretical results in fact boil down to algorithms and techniques that work in practice. The goal of this project is to derive a proof-of-concept implementation of approaches based on weighted automata.

1 Motivation and Overview

Model checking discrete event dynamic systems is an area that combines a variety of types of automata with a variety of types of modal logics to evaluate if a given system has a particular property or not. Those automata models differ in important aspects; for instance untimed automata are used for standard model checking, probabilistic automata for timed and probabilistic model checking, stochastic automata for stochastic model checking and various other forms of timed automata have been considered as well. By considering the wide area of finite state automata, one can notice that in addition to these types of automata other models have been proposed and applied successfully in different application areas. Examples are min/plus, max/plus, or min/max automata that have been used for the analysis of real time systems, communication system, and discrete event systems. Furthermore, similar models have been applied for natural language processing or image compression. It is quite natural and for most of the mentioned applications also very useful to extend model checking approaches to all these types of automata. Since the class of weighted automata provides in some sense a superset of different automata types, which includes different forms of probabilistic automata and also untimed automata, one may strive for a general framework of model checking which can be applied to a wide variety of different types of weighted automata without defining a new approach for each type. Such a framework is of theoretical interest to get a better understanding of model checking and to get a common ground for model checking in various application areas. From a methodological point of view, it gives direct access to model checking techniques for various types of automata that do not profit from these techniques yet. Finally, it supports tool development: in an object oriented setting, implementation of a specific model checker can inherit basic techniques from a more general class that implements techniques valid for the whole framework.

Weighted automata are a well known class of automata where transitions are labeled with labels from a finite alphabet and, additionally, receive weights or costs that are elements of some semiring. A key observation is that the algebraic structure of a semiring is sufficient to define model checking for weighted automata. The advantage is that by selecting appropriate semirings, one obtains different types of automata that include most of the above mentioned types. This general type of automata is suitable to define a bisimulation as we did in [4, 2]. In [3], the process algebra GPA has been introduced for the specification of models in a compositional way such that the underlying semantic model is a weighted automaton in the case of a finite set of states. In [1] we propose a model checking approach

for weighted automata. The approach allows us to check formulas of the newly defined logic *Valued Computational Tree Logic* (CTL$) over a weighted automaton. Algorithms for model checking are developed and it will be shown that by an appropriate definition of the semiring used for the definition of transitions weights, we naturally define model checking approaches for different model types without developing new approaches in each case. The special cases include untimed, probabilistic, min/plus, max/plus, and min/max automata such that known model checking approaches are covered and new approaches are introduced in the case of min/plus, max/plus, and min/max automata. By the use of other semirings for transition weights, the proposed approach applies to a wide class of automata models. In so far, we develop some form of a generic approach for model checking that is applicable to other model classes and that includes algorithms to perform model checking.

Schedule and Management plan "The proof of the pudding is in the eating" is the motto of this project, which aims at developing a proof-of-concept implementation of a number of concepts known for weighted automata. Before we discuss a schedule with milestones, let's consider a number of issues that formulate challenges and possible solutions for our endeavor.

- The implementation should be generic in the sense that the underlying semiring can be exchanged without any other changes to the code base. This could be derived by working with a semiring interface that specific semirings implement and the prototype pattern (e.g. as discussed in Horstmann).
- Automata come with a number of composition operations, such that bisimulations turn out to be congruences. A compositional structure is known to provide valuable information for state space exploration (symbolic or Kronecker methods) as well as for bisimulation based reduction operations. Classes that represent automata may retain and take advantage of information on a given compositional structure and may use the Composite pattern (e.g. as discussed in Horstmann).
- Subtle yet important differences between semirings, automata, representations should be indicated by attributes or resolved by polymorphism.
- State space exploration and state-based methods have simple straightforward implementations and sophisticated ones based on symbolic or Kronecker representations. A symbolic exploration has to rely on an existing library to be feasible within the given time frame, hence we should start of with a simple implementation and have the state space exploration and its outcome be properly encapsulated such that it is simple to replace a straightforward algorithm by a symbolic one.
- Modeling and analysis techniques consider conceptually the same discrete event dynamic system yet from different perspectives. In Mobius, this is resolved by a so-called model-level abstract functional interface and by a state-level abstract functional interface (model-level and state-level AFI). While the former encapsulates the representation of a state and state transition function of a model, the latter allows an analysis technique to consider a DEDS basically as a transition matrix. A similar distinction may be useful for this project as well.
- Team effort vs individual grades. Obviously the overall project is more interesting if we investigate and implement different aspects and not redundant ones. In order to allow for individual grades, subprojects will be assigned to the sole responsibility of a single person and will be evaluated and graded within its own test environment. However, as an incentive towards the overall bonus points given for an overall working solution.

We make use of a test driven design with several iteration phases. The idea is to start of from a basic implementation that has default implementations for particular examples such that only limited functionality is in fact available and which will grow over time and phases.

- phase 1 (Feb 19- Feb 26, 1 week): class design with CRC cards, definition of model-level and state-level AFI, implementing a test suite for a use case and trivial default implementation, phase is finished if overall design is consistent with respect to interfaces and required functionality.
- phase 2 (Feb 26 - March 11, 2 weeks): extending the test suite for one example taken from the literature and a first implementation that handles at least the example from the literature. Phase is finished if overall implementation handles the example.
- phase 3 (March 11 - March 25, 2 weeks): extending the test suite to cover further examples that use more features, full fledged implementation, phase ends if overall implementation handles set of well-specified correct examples that exercise full functionality.
- phase 4 (March 25 - April 8): extending the test suite for use cases that check exceptional cases, phase ends if overall implementation is able to handle complete set of examples in a reasonable manner.
- phase 5 (April 8 - April 22): wrap up, evaluation and documentation phase. In this phase, the result of phase 4 is evaluated with respect to its potential and limitations. Final results include an evaluation report, a documented and tested code base with a set of examples.

In what follows, we break down the overall project into five individual tasks.

Task 1 is to obtain a weighted automaton from a simulation trace. The outcome of this task will help us to obtain application examples for subsequent analysis in a convenient and automated manner.

Task 2 is to obtain a weighted automaton from a general process algebra. Similar to task 1, the outcome will make the overall project more useful and easier to evaluate.

Task 3 is to transform a model-level description (implementation of the model-level AFI) into an implementation of the state-level AFI. This task requires a state space exploration and is fundamental for any state based analysis.

Task 4 is to support the transformation in task 3 by a bisimulation minimization. The outcome helps to make the overall approach scale further.

Task 5 is to provide modelchecking functionality based on the state-level AFI such that we can evaluate models for terms of an appropriate modal logic like CTL$.

In what follows, we describe these tasks in further detail.

2 Task 1: Implementing a model-level interface: Automata derived from traces

A trace σ is an alternating sequence of states and transitions that may result from monitoring a system or from simulating a discrete event dynamic system. We can derive traces from simulators like Mobius and can make use of a trace parser (in Java) from the existing Traviando trace analyzer.

The model level AFI requires us to implement a weighted automaton that has an internal notion of state and state transitions. We need to provide an initial state, a state transition relation, access to all enabled transitions for a given state and an equality and an equivalence relation for states. For the state transition relation, we need to produce a successor state for a given transition label (if enabled) as well as a weight for that transition. We also need to provide a prototype implementation of the semiring, that is used for this model. For a start, we use the (R,+,x,0,1) semiring, we are familiar with.

If the model is compositional, we need to identify the type of composition (direct product, synchronized product, choice, sum of automata), we need to give access to the constituents as well as to the set of shared labels in case of a synchronized product.

Traviando traces can be seen as a synchronized product of weighted automata in a natural manner. A traviando trace has a prefix that identifies a number of processes that have individual state variables and individual (local) actions as well as a set of actions that are shared and used between processes for

synchronisation. Since the state of each process can be identified by the settings of its state variables and state changes by the occurrence of corresponding actions in σ, we can create one automaton for each process with as many states as there are different local states for that trace in σ. For any action that takes place with respect to a particular automaton, we can check the difference of time stamps to the previous action (of that process) and take those numerical values as weights. If several actions with same label and to the same successor state are observed for a particular state, where the timing is different, we represent that set of values by its mean, minimum and maximum value. For synchronized actions, we need to keep track of the most recent predecessor action to get the relevant timing information. Note that the language of the resulting automata is richer than just σ, it includes σ as one element but is likely o contain many more.

Other derivations of automata are possible as well, e.g. a trivial one that creates a single automaton that generates that particular finite sequence σ. One may consider the Sequitur algorithm as one way to derive a concise internal representation of that trace. Furthermore, one can adopt the CTMC generation algorithm of Sen, Viswanathan and Agha to obtain a weighted automaton from a set of traces.

Side issues: Traviando traces are available in an XML format, a parser that reads a given trace into a Java class "trace" is available. Interface and implementation of semirings following the prototype pattern (e.g. as presented by Horstmann) is shared with other tasks. The notion of equivalence should support bisimulations considered in Task 4.

3 Task 2: Implementing a model-level interface: Automata derived from a Process Algebra

For this task, we need to parse a GPA model description, presumedly from a yet to be defined XML format. With that information, we need to implement the model-level AFI, which means that we design and implement a notion of state and a state-transition relation based on the grammatical rules defined for the GPA in [3].

The model level AFI requires us to implement a weighted automaton that has an internal notion of state and state transitions. We need to provide an initial state, a state transition relation, access to all enabled transitions for a given state and an equality and an equivalence relation for states. For the state transition relation, we need to produce a successor state for a given transition label (if enabled) as well as a weight for that transition. We also need to provide a prototype implementation of the semiring, that is used for this model. For a start, we use the (R,+,x,0,1) semiring, we are familiar with.

If the model is compositional, we need to identify the type of composition (direct product, synchronized product, choice, sum of automata), we need to give access to the constituents as well as to the set of shared labels in case of a synchronized product. For GPA, it is expected to use the synchronized product to come up with a compositional model.

Side issues: Interface and implementation of semirings following the prototype pattern (e.g. as presented by Horstmann) is shared with other tasks. The notion of equivalence should support bisimulations considered in Task 4.

4 Task 3: State space exploration

For this task, we need to adopt one of the known state space exploration algorithms. For an initial trivial solution, one can derive a BFS-algorithm as performed for symbolic exploration but in fact use simple set data structures and explore one transition and state after the other. Subsequent versions may then make use of a decision diagram library like libddd to proceed further. The challenge for a symbolic exploration is in the generation of a suitable encoding of the next state relation based on what is provided by the model-level AFI. There is related work by Siegle and Lampka to do so.

The outcome of this task is an implementation of the state-level AFI, that allows us to access a model like a state transition matrix, i.e., we can identify its dimension and obtain state=transitions based on ids. If a symbolic exploration is achieved, we can generalize the interface to handle sets of states instead of single states. Note that this transformation must also transform the evaluation of atomic propositions as labels of the state space S in order to support model checking.

5 Task 4: Bisimulation minimization

For this task, we need to implement a partition refinement algorithm that serves various semirings (one implementation should suffice), various types of bisimulations (one implementation should suffice, or minor variations that exploit inheritance). The outcome should be a data structure in close cooperation with task 3 such that the state space exploration in fact can take advantage of a compositional minimization of a model. For a given model, a set of atomic propositions, a particular bisimulation, this tasks produces a minimal equivalent weighted automaton that implements the same interfaces as the input and can safely replace the input model for any further analysis. The various and closely related kinds of bisimulations are discussed in [2], [1] Section 5, [3] Section 4, and as forward and backward bisimulationsin [4]. For algorithms, check references [7,10] given in [2] and work by S. Derisavi and H. Herrmanns on this topic.

6 Task 5: Modelchecking

For this task, we need to implement modelchecking algorithms described in [**?**] for the modal logic CTL$ based on the state-level AFI. The connection to the particular modeling formalism is based on atomic propositions - the usual formal trick seen in the literature to separate modal logics and model checking from particularities of a modeling formalism. Atomic propositions are boolean functions that evaluate for a given state to true or false. This implies requirements for a model-level AFI to support atomic propositions and for state space exploration and bisimulation minimizations to preserve those propositions and to make those accessible via the state-level AFI.

The logic CTL$ has mainly two non-trivial path operators, U and AU that require considerations of paths in a weighted automaton.

7 Examples

A number of examples can be adopted from the various papers on weighted automata, for instance, see Section 6 in [2], the driving test example in [1], page 4 and Section 6 in [1] and Section 5 in [3].

References

[1] P. Buchholz, P. Kemper. Model Checking for Automata with Transition Costs. submitted for publication.

[2] Buchholz,P. and Kemper,P. Weak bisimulation for (max/+) automata and related models Journal of Automata, Languages and Combinatorics, selected papers of the workshop Weighted Automata: Theory and Applications (Dresden, Germany, March 4-8, 2002), Vol. 8 (2003) Number 2.

[3] Buchholz,P; Kemper,P. Quantifying the Dynamic Behavior of Process Algebras In Proc. of the joint PAPM-PROBMIV workshop, September, 2001, Aachen, Germany, Springer, LNCS 2165, pp. 184-199, 2001.

[4] P. Buchholz. Bisimulation Relations for Weighted Automata. To appear in Theoretical Computer Science.