The Mec 5 model-checker

Alain Griffault and Aymeric Vincent[⋆]

LaBRI, Bordeaux University, 351 cours de la Libération, 33405 Talence, France
`<Firstname.Surname@labri.fr>`

Abstract. We present in this article the features of the model-checker we have developed: Mec 5. This tool makes it possible to handle models written in the AltaRica language and is characterized by the great expressiveness of its specification logic: μ-calculus on relations with first order quantifiers and equality.

Keywords: AltaRica, BDDs, model-checking, μ-calculus

1 Introduction

Mec 5 [10] is a model-checker for finite AltaRica [9, 2] models. The way to specify properties in Mec 5 consists in defining relations in a specification language we describe in section 2. In this setting, one can verify that a system satisfies a given property by testing that the set of states which violate the property is empty. This specification language is very expressive and allows the definition of complex relations between different models, e.g. bisimulation. Mec 5 is also open to more general problems like controller synthesis because it can compute winning strategies in parity games.

AltaRica is a rich formalism developed at the LaBRI jointly with industrial partners. Its goal is to provide a formalism with clear semantics, supported by a language offering powerful modeling facilities, itself supported by many tools in order to perform safety analysis of a given AltaRica model using several different techniques. Compilers exist to produce fault trees, Markov chains, Lustre programs, transitions systems.

The need for an AltaRica model-checker was real because the availability of commercial tools to model in AltaRica brings libraries of components whose correctness must be verified.

2 Specification language

The specification language used by Mec 5 coincides exactly with Park's μ-calculus [8] which is first order logic extended with fixpoints over relations.

First-order logic

The propositions are built using the traditional boolean connectives (`~` for negation, `&` for conjunction, `|` for disjunction). Variables range over finite domains like booleans (`bool`), finite intervals of integers (`[0, 10]`) or enumerations of constants (`{on, off}`).

[⋆] currently a post-doc at the University of Warsaw in the European RTN "GAMES"

A relation can be used in this context as a predicate, i.e. like the characteristic function of the relation it represents, returning a boolean value (`R(x, 2)`).

The introduction of first-order quantifiers in our language allows us to dispose of the existential and universal modalities "*for all successors...*" and "*there exists a successor such that...*" whose semantics are usually given explicitly and which are usually the only link between the specification language and the model under study. We use a concrete syntax which reminds those modalities: $\exists x.p$ is written `<x>p` and $\forall x.p$ is written `[x]p`.

Fixpoints over relations

Given a monotonic function over relations, it is possible to compute the least (`+=`) or the greatest (`-=`) relation which is a fixpoint of this function. For example, computing the transitive closure `T` of a relation `R` can be written verbatim like this in Mec 5 using a least fixpoint:

```
T(x, y) += R(x, y) | <z>(R(x, z) & T(z, y));
```

Properties depending on several models are easy to express. Assuming an equivalence relation `eq(a,b)` on labels, bisimulation can be written like this:

```
bisim(s,s') -=
  ([e][t](R(s,e,t)=><e'><t'>(R'(s',e',t')&eq(e,e')&bisim(t,t'))))&
  ([e'][t'](R'(s',e',t')=><e><t>(R(s,e,t)&eq(e,e')&bisim(t,t'))));
```

It is also possible to use several interdependant fixpoint definitions by means of systems of equations [6], which gives Mec 5 the full power of μ-calculus.

Implementation

Mec 5 uses Binary Decision Diagrams [3] to represent relations. This choice fits exactly our specification language, because boolean as well as first order operations are efficient on BDDs. The test for equality can be done in constant time, which is valuable when computing fixpoints. Expressions which range over finite domains are handled with vectors of BDDs. As a side note, our implementation benefits currently from a few performance improvements: BDDs use negative edges so that negation can be computed in constant time, and the boolean variables used to encode a relation are interleaved and we are working on wiser heuristics to reduce the memory footprint.

3 Integration with AltaRica

The AltaRica formalism is based on constraint automata and provides facilities to allow the modeling of complex systems. An AltaRica *node* can be defined hierarchically, by first modeling small parts of the system (themselves nodes) and then gathering them as sub-nodes. The default semantics for sub-nodes is to run asynchronously, unless some events are explicitly synchronized. Indeed, two means of communication are provided:

Memory sharing The *assertion* of a node can relate local variables to variables of its sub-nodes and can be seen as an invariant of the node. By forcing two variables to be equal, it is easy to share information between nodes; more complex constraints can be used. `(A.f1 = B.f) & (A.f2 < C.f)`

Events synchronization Synchronization vectors `<A.a, B.b>` specify which events should be bundled together, preventing them from occurring independently, and allowing a transition to fire all the specified events in parallel.

Other mechanisms provided in AltaRica include the ability to specify priority constraints on events `a<b` as a partial order. *Broadcast vectors* `<a,b?>` extend synchronization vectors by allowing certain events (those with a `?` mark) not to occur while maximizing the number of fired events: `<a,b?>` ≡ `{<a>} < {<a,b>}`.

The tight coupling between the AltaRica description language and Mec's specification language was one of our primary goals due to our need for an efficient model-checker in the AltaRica community. This was done by using the same basic types in the two languages (booleans, intervals and enumerations), and using the same concrete syntax wherever possible for the definition of new domains, for expressions and for type expressions. It makes Mec 5 a coherent tool.

Every AltaRica model `A` loaded in Mec 5 defines two types and two relations : the type of its configurations `A!c`, the type of its event vectors `A!ev`, its transition relation `A!t`⊆`A!c`×`A!ev`×`A!c` and its set of initial configurations `A!init`⊆`A!c` which is the set of all configurations by default. Given these new objects, any property on `A` can be expressed in Mec 5's specification language.

4 Example

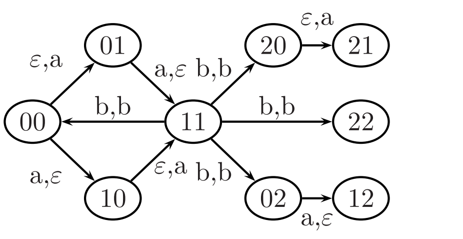
We give a very simple AltaRica model which is a loop from which it is possible to escape non-deterministically; we define the product of two such loops and then compute the set of states from which it is unavoidable to go to a dead state:

```
NotEpsilon(e : main!ev) := ~(e.S1. = "" & e.S2. = "");
UnavDead(c) +=
    [e][c'](main!t(c, e, c') & NotEpsilon(e)) => UnavDead(c'));
```

```
node LoopExit
    state s : [0, 2];
    event a, b;
    trans s = 0 |- a -> s := 1;
          s = 1 |- b -> s := 0;
          s = 1 |- b -> s := 2;
edon

node main
    sub S1, S2 : LoopExit;
    sync <S1.b, S2.b>;
edon
```

AltaRica description



Semantics of `main` without ε loops

```
({S1.s = 2, S2.s = 2})
({S1.s = 1, S2.s = 2})
({S1.s = 0, S2.s = 2})
({S1.s = 2, S2.s = 1})
({S1.s = 2, S2.s = 0})
```

Result

5 Conclusion

Mec 5 currently uses common techniques like Binary Decision Diagrams, and in that sense is very similar to tools like SMV [7] or NuSMV [4]. However, it departs from these model-checkers because it provides a more powerful logic that is not specifically designed for model-checking. In this respect, Mec 5 is closely related to Toupie [5] which implements Park's μ-calculus with decision diagrams but does not provide the means to load a model.

The experiments we made show that the specification language, although unusual at first glance, is extremely versatile and expressing properties with it is quite easy. Mec 5 and examples of AltaRica models are available from the following URL: `http://altarica.labri.fr/`

The ongoing evolution of Mec 5 follows two lines: performance improvement and implementation of satellite facilities to help in the verification process (either in the core of the tool, e.g. manipulating traces of executions, or in a graphical interface, e.g. a simulator). We expect to improve performance by using the knowledge we can extract from an AltaRica model to choose a good variable ordering for the BDDs, and we are investigating more symbolic methods which would delay the computation of BDDs, in the spirit of what was done with Boolean Expression Diagrams [1].

References

1. Henrik Andersen and Henrik Hulgaard. Boolean expression diagrams. *Information and Computation*, 179(2):194–212, December 2002.
2. André Arnold, Alain Griffault, Gérald Point, and Antoine Rauzy. The altarica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40(2–3):109–124, 1999.
3. Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
4. Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV version 2: An opensource tool for symbolic model checking. In *CAV: International Conference on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, July 2002.
5. Marc-Michel Corsini and Antoine Rauzy. Toupie user's manual. Research Report 586-93, LaBRI, 1993.
6. Angelika Mader. *Verification of modal properties using boolean equation systems.* PhD thesis, Fakultät Informatik, Technische Universität München, 1997.
7. Kenneth L. McMillan. *Symbolic Model Checking: an approach to the state explosion problem.* PhD thesis, Carnegie Mellon University, May 1992.
8. David Park. Finiteness is μ-ineffable. *Theoretical Computer Science*, 3:173–181, 1976.
9. Gérald Point. *Altarica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement.* PhD thesis, LaBRI, Université Bordeaux 1, January 2000.
10. Aymeric Vincent. *Conception et réalisation d'un vérificateur de modèles AltaRica.* PhD thesis, LaBRI, Université Bordeaux 1, December 2003.