Merging DBMs Efficiently – Extended Abstract

Alexandre David[1]

Department of Computer Science, Aalborg University, Denmark
{adavid,kgl}@cs.auc.dk.

Abstract. In this paper we present different algorithms to reduce the number of DBMs in federations by merging them. Federations are unions of DBMs and are used to represent non-convex zones. Inclusion checking between DBMs is a limited technique to reduce the size of federations and how to choose some DBMs to merge them into a larger one is a combinatorial problem. We present a number of simple but efficient techniques to avoid searching the combinations while still being able to merge any number of DBMs.

1 Introduction

DBMs (difference bound matrices) [7, 6] are efficient data structure to represent clock constraints in timed automata [1]. However, some operations require splitting, e.g. substraction or some extrapolation algorithms [2] because DBMs can not represent non-convex zones. Such unions of DBMs, called federations, are subject to further operations to compute successor states in reachability algorithms and they may be split again. Keeping control of the size of a federation (its number of DBMs) is therefor vital. Removing included DBMs is a limited technique with respect to splitting. Our algorithms are able to merge adjacent DBMs.

The new DBM library of the model-checker UPPAAL[1] supports federations and all the operations usually carried out on DBMs. In addition it supports substractions and our merging algorithms. There are other representations of zones that can deal with non-convex zones, such as CDDs [5] or CRDs [8]. In this paper we are concerned about how to deal with DBMs as best as we can since DBMs are already used in our model-checker. Depending on the representation chosen, an operation may be more or less efficient, we do not address this issue.

2 Notations.

DBMs are used to represent symbolically sets of clock valuations in timed automata. The representation is given as a matrix of constraints $x_i - x_j \sim b_{ij}$ where $\sim \in \{<, \leq\}$, $1 \leq i, j \leq n$, x_i and x_j are clock valuations of the clocks i and j, and b_{ij} are the bounds of the constraints. A *zone* corresponds to the set of clock valuations that satisfy the constraints of a given DBM. Zones are convex by definition of DBMs.

[1] http://www.uppaal.com

Definition 1 (Comparison of constraints) For c_{ij} and z_{ij} constraints we define:

- $c_{ij} \leq z_{ij} \Leftrightarrow c_{ij} \Rightarrow z_{ij}$
- $c_{ij} < z_{ij} \Leftrightarrow c_{ij} \neq z_{ij} \wedge c_{ij} \leq z_{ij}$
- $-c_{ij} = -(x_i - x_j \sim b_{ij}) = x_j - x_i \bar{\sim} - b_{ij}$, where $\bar{\leq} =<$ and $\bar{<} =\leq$. Note that $-c_{ij}$ is a new c'_{ji} comparable with other constraints z_{ji}.
- $(x_i - x_j < b_{ij})^{+} = (x_i - x_j \leq b_{ij})^{+} = (x_i - x_j \leq b_{ij})$.

A DBM is *canonical* if its constraints have been tightened by Floyd's shortest path algorithm [4]. We consider that DBMs are always canonical. We always use c_{ij}, respectively z_{ij}, to denote the constraints of the DBM C, respectively Z. A union of DBMs (federation) represents the set of clock valuations that satisfy the constraints of any such DBM.

3 The Combinatorial Problem

The basic idea in merging a union Z of k DBMS Z_i is to compute a cheap superset of the union with the convex hull C_Z of Z (we have $Z \subseteq C_Z$ by definition) and to check if $C_Z \subseteq Z$. The inclusion is computed by checking if $C_Z - Z = \emptyset$. If the inclusion holds then the convex hull is equal to the union of the DBMs Z_i and we can replace Z by C_Z. The whole problem for a given federation is to find such a subset Z to merge its DBMs. If the original federation has n DBMs, there are $\binom{n}{k}$ ways to try to merge them for a given k. Trying all k gives a total of 2^n combinations. Every attempt is expensive, which makes this procedure infeasible in practice.

4 Our Heuristics

We present different heuristics to avoid the combinatorial problem of choosing DBMs. We test only n^2 combinations with greedy criteria to gather two or more DBMs to be merged.

2-Merge We check all DBMs by pairs and if two DBMs A and B satisfy the necessary (but not sufficient) criteria:

1. $\exists i, j.\ a_{ij} = b_{ij}\ \wedge\ a_{ji} = b_{ji}$.
2. $\forall i, j.\ \neg(-a_{ij}^{+} \geq b_{ji}^{+}\ \vee\ -a_{ji}^{+} \geq bij^{+})$.

We attempt the merge with the convex hull as explained in Section 3. The criteria check for compatible opposite constraints and if the relaxed DBMs do not intersect. This cheap test allows us to try only on DBMs that have a good chance to be merged. It is very cheap to add inclusion checking in the same algorithm to eliminate included DBMs on-the-fly.

n-Merge This is a relaxed version of the 2-merge algorithm where the first condition is $\exists i, j.\ a_{ij} = b_{ij} \vee a_{ji} = b_{ji}$, in which case we gather B in a federation F_B of DBMs to be merged with A. In addition, we add any DBM that is included in the convex hull of A and F_B denoted $ConvexHull(A, F_B)$. This is to "fill the holes" if possible. Then we check if $R = ConvexHull(A, F_B) - (A \cup F_B)$ is empty, in which case the merge is successful. Otherwise we check if $ConvexHull(A, F_B) - R$ gives a shorter federation than A and F_B, in which case we can recompute the federation with fewer DBMs. If this still fails, we apply a stronger inclusion checking to the DBMs of F_B (based on subtraction) since we have the information that they are related. This algorithm performs well in practice and is able to merge sets of n DBMs while avoiding any combinatorial problem.

Partitioned n-Merge This is a refinement of the $n - merge$ algorithm: We detect partitions inside the original federation (disjoint sets of relaxed DBMs) and we apply the $n - merge$ algorithm followed by a strong inclusion check (based on subtraction) on them.

5 Conclusion

The algorithms we propose have been implemented in our DBM library (distributed under the GPL license). We developed these algorithms originally for a prototype to solve timed games [3] because the federations exploded. Our heuristics have kept the size of the federations in control and have proven to be very useful in practice.

References

1. Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *Proc. of Int. Colloquium on Algorithms, Languages, and Programming*, volume 443 of *LNCS*, pages 322–335, 1990.
2. Johan Bengtsson. *Clocks, DBMs and States in Timed Systems.* PhD thesis, Uppsala University, 2002.
3. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR'05*, volume 3653 of *LNCS*, pages 66–80. Springer–Verlag, August 2005.
4. Robert W. Floyd. Acm algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.
5. Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.
6. Kim G. Larsen, Paul Pettersson, and Wang 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, August 1995.
7. Tomas Gerhard Rokicki. *Representing and Modeling Digital Circuits.* PhD thesis, Stanford University, 1993.
8. Farn Wang. RED: Model-checker for timed automata with clock-restriction diagram. In Paul Pettersson and Sergio Yovine, editors, *Workshop on Real-Time Tools, Aalborg University Denmark*, number 2001-014 in Technical Report. Uppsala University, 2001.