Minimal DBM Substraction

Alexandre David[1], Johan Håkansson[2], Kim G. Larsen[1], and Paul Pettersson[2]

[1] Department of Computer Science, Aalborg University, Denmark
{adavid,kgl}@cs.auc.dk.
[2] Department of Information Technology, Uppsala University, Sweden
{johnh,paupet}@it.uu.se.

Abstract. In this paper we present an algorithm to compute DBM substractions with a guaranteed minimal number of splits and disjoint DBMs to avoid any redundance. The substraction is one of the few operations that result in a non-convex zone, and thus, requires splitting. It is of prime importance to reduce the number of generated DBMs in an exploration loop, e.g., for reachability, because the result is propagated and serves to compute further successors later.

1   Introduction

DBMs (difference bound matrices) [6, 4] 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. The resulting DBMs of a splitting are parts of successor states that will be used to compute further successors. Reducing splitting means to reduce the state explosion.

The new DBM library of the model-checker UPPAAL[1] supports substractions and federations of DBMs to represent non-convex zones. There are other representations of zones that can deal with non-convex zones, such as CDDs [3] or CRDs [7]. In this paper we are concerned about how to solve the substraction for the DBMs when the DBMs are already used in a model-checker. Depending on the operation, a given representation may be more or less efficient, we do not address this issue.

2   DBM Substraction

Given two DBMs A and S, we want to substract S from A. The resulting zone Z can be defined as the zone satisfying the constraints of A and $\neg S$. Intuitively, a point $p \in \neg S$ iff $\neg(p \in S)$. Computing the result $Z = A \wedge \neg S$ is done by constraining A with the negated constraints of S. Figure 1 illustrates the basic substraction algorithm in two dimensions, i.e., with two clocks. In the worst case, for a DBM of dimension n, there are n^2 splits where each split costs a copy. The algorithm complexity is $O(n^4)$.
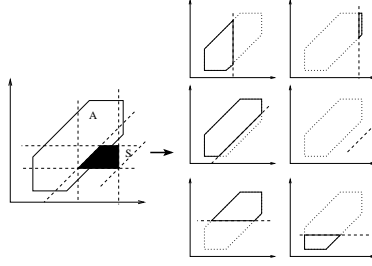
---

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

Fig. 1. Basic substraction algorithm.

3 Reducing Splitting

The first observation from the basic algorithm is that some splits may be avoided by taking into account only the edges that belong to the minimal graph [5] of the DBM. A DBM can be seen as a directed graph between clocks with the constraints on the edges. Figure 2 shows the reduced substraction by using only these constraints. The complexity for computing the minimal graph is $O(n^3)$ but it is worth doing since it is more important to reduce the result.
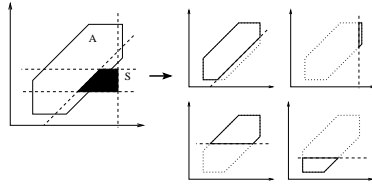

Fig. 2. Substraction using the constraints part of the minimal graph.

4 Substraction with Minimal Split

To further reduce the number of splits, let us consider the following four cases that arise on the constraints belonging to the minimal graph of the DBM to substract:

1. The (negated) constraint reduces A to an empty zone: we ignore it.
2. The (negated) constraint has no effect on A: because A is convex, this means that the DBM S to substract is outside of A so we stop and the result is A.
3. The (negated) constraint is on a facet of S that does not intersect A: we ignore it.
4. Otherwise we compute a split.

The third case is the contribution of our algorithm: the decision procedure is linear in the number of clocks and can rule out constraints that do not affect the result. The main argument for ruling out these constraints comes from the convexity of A and S: if a facet of S does not intersect A, then it has no effect on the substraction. We argue that this algorithm is sound and complete. As a remark, the first case is redundant with the third case but it is a simple test used to rule out simple cases. The complexity of the additional test is $O(n)$ per constraint, which is, $O(n^3)$ in total: we do not make the substraction worse.
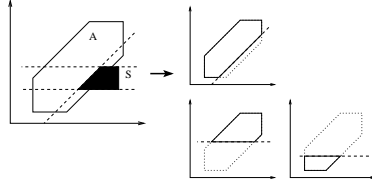

Fig. 3. Substraction with minimal splitting.

5 Minimal Substraction

Having the minimal number of DBMs as the result from a substraction may not be enough: if the successor states are going to be arguments for further substractions then there should be no overlapping between them otherwise future substractions will be redundant. The bad thing could be that future generated DBMs may not be simply comparable with respect to inclusion checking, which means, that even if some of them are redundant, they will be kept. We propose a simple procedure to reduce the size of the resulting DBMs to make them disjoint. The ordering of the splitting procedure affects the result but it is still guaranteed to be minimal and disjoint. This procedure costs a copy per split. The complexity is $O(n^4)$ in total. The substraction is still in $O(n^4)$.
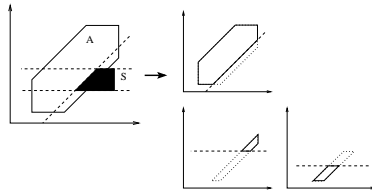

Fig. 4. Substraction with minimal splitting and disjoint result.

6 Conclusion

The algorithm we propose has been implemented and tested with other functions of the DBM library that use substraction as a sub-function. The library has an extensive set of tests that can be used as benchmarks. It turns out that it is more important to have a reduced result rather than a cheap and redundant one. The reason comes from the propagation of the result in the exploration loop of the model-checker, e.g., for reachability, or in the tests we made on the propagation of partial results: the resulting DBMs are used later to compute further successors and substractions.

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. Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.
4. 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.
5. Fredrik Larsson, Kim G. Larsen, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society Press, December 1997.
6. Tomas Gerhard Rokicki. *Representing and Modeling Digital Circuits*. PhD thesis, Stanford University, 1993.
7. 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.