Performance Investigation of a Subset-Tuple Büchi Complementation Construction

Daniel Weibel

November 1, 2014

Contents

Chapter 1

The Fribourg Construction

The Fribourg construction draws from several ideas: the subset construction, run analysis based on reduced split trees, and Kurshan's construction [8] for complementing DBW. Following the classification we used in Section **??**, it is a slice-based construction. Some of its formalisations are similar to the slice-based construction by Vardi and Wilke [25], however, the Fribourg construction has been developed independently. Furthermore, as we will see in Chapter **??**, the empirical performance of Vardi and Wilke's construction and the Fribourg construction differ considerably, in favour of the latter.

Basically, the Fribourg construction proceeds in two stages. First it constructs the so-called upper part of the complement automaton, and then adds to it its so-called lower part. These terms stem from the fact that it is often convenient to draw the lower part below the previously drawn upper part. The partitioning in these two parts is inspired by Kurshan's complementation construction for DBW. The upper part of the Fribourg construction contains no accepting states and is intended to model the finite "start phase" of a run. At every state of the upper part, a run has the non-deterministic choice to either stay in the upper part or to move to the lower part. Once in the lower part, a run must stay there forever (or until it ends if it is discontinued). That is, the lower part models the infinite "after-start phase" of a run. The lower part now includes accepting states in a sophisticated way so that at least one run on word w will be accepted if and only if all the runs of the input NBW on w are rejected.

As it may be apparent from this short summary, the construction of the lower part is much more involved than the construction of the upper part.

1.0.1 Construction of the Upper Part

The construction of the upper part takes as input a NBW A and outputs a deterministic automaton B' that will be the upper part of the final complement automaton B. The construction of B' is in its approach similar to the subset construction. One starts with a B'-state representing the initial state of A and then recursively determines and adds for each B'-state one successor state per input symbol. The difference to the subset construction lies in the inner structure of the B'-states. In the subset construction this would simply be a single set of A-states. In the subset-tuple construction, however, a B'-state is a tuple of sets of A-states. A tuple is an ordered list, so differently phrased, a B'-state consists of one or more sets of A-states where the order of these sets matters. As we will see, the A-states present in a B'-state are the same that would result from the subset construction. But while in the subset construction all these states are thrown together in one set, in the subset-tuple construction they are split up in multiple sets where additionally the order of these sets is important. The subset-tuple construction can thus be seen as a modified subset construction that includes additional structure.

The structure of B'-states is defined by "level-clippings" of reduced split trees, as we explain in a moment. But, talking about reduced split trees, we have to make a decision first. In Section **??** we mentioned that reduced split trees come in equivalent left-to-right and right-to-left versions.

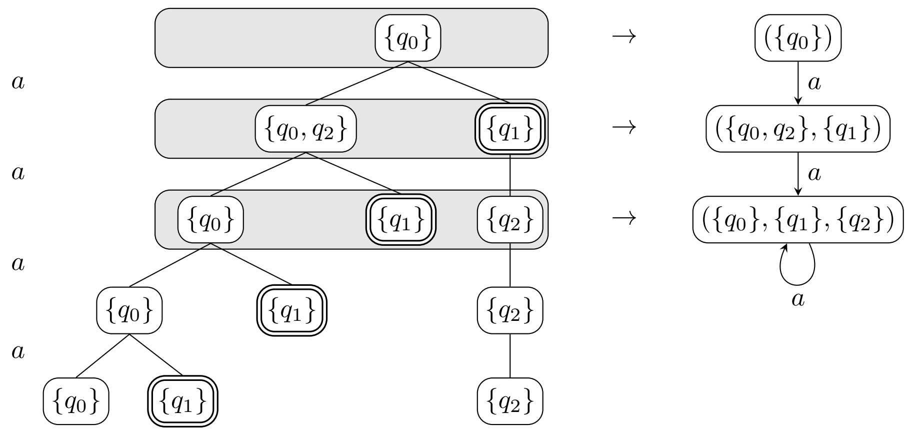

Figure 1.1: From levels of a reduced split tree to the slices of the subset-tuple construction.

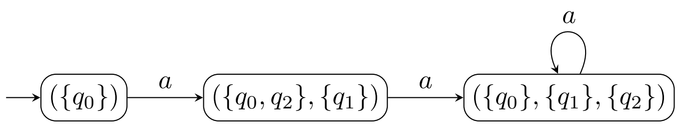

Figure 1.2: Upper part of complement of A.

The construction has to adopt one of these variants and stick to it. In this thesis we use the right-to-left version. The final complement automata look the same with either version except that the order of the tuples is reversed. Note that the optimisations described in Section **??** is also based on this ordering and would need to be rephrased for the left-to-right version.

A new B'-state q, successor on symbol a of an already existing B'-state p, now is created in the following way. The predecessor p is regarded as a level of a reduced split tree by looking at its sets as the nodes on this level. Then, the next level for the given symbol a is constructed as described in Section **??**. This new level then directly defines q by taking the nodes as the sets of q's tuple and keeping their order. Figure **??** illustrates this with the example automaton A that we already used before. If a state with a simlar tuple to the just created q already exists in B', then just a transition from p to this state is added (hence the loop in the third state of Figure **??**). The construction starts with a B'-state containing only A's initial state and ends when all states of B' have been processed for all input symbols. In Figure **??**, the construction is complete, thus the automaton shown at the right is the upper part of the complement of A.

If all the A-states of a B'-state p have no successors on an input symbol a, then p will have no a-successor in B'. This results in the upper part B' being incomplete at the end of the construction. In this case, it has to be made complete by adding a sink state. Furthermore, this sink state has to be accepting.

States of the subset-tuple construction are thus levels of reduced split trees. In the constructions of Varid and Wilke [25], and Kähler and Wilke [5] states are called *slices* of reduced split trees, hence the name slice-based approach.

1.0.2 Construction of the Lower Part

The construction of the lower part takes as input the upper part B' (and the initial automaton A) and outputs the final complement automaton B with $L(B) = \text{L(A)}$. The construction of the lower part is basically an extension of the construction of the upper part that is applied to the states of the upper part. The extension consists therein that every set in the states of the lower part is assigned a *colour*. These colours will be used to keep track of certain properties of runs of B that

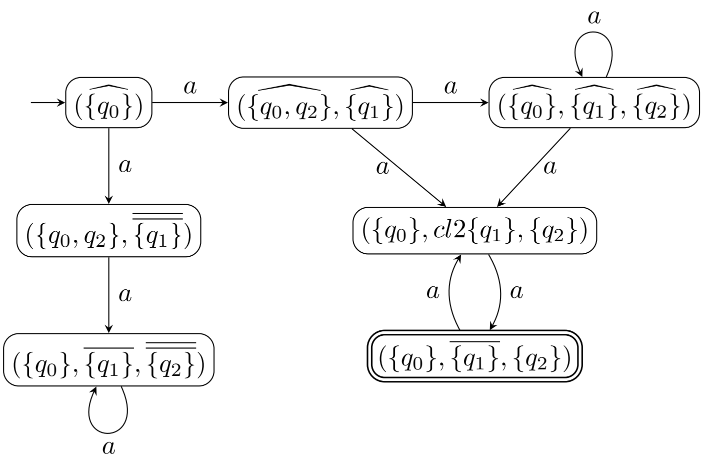

Figure 1.3: The final complement automaton B.

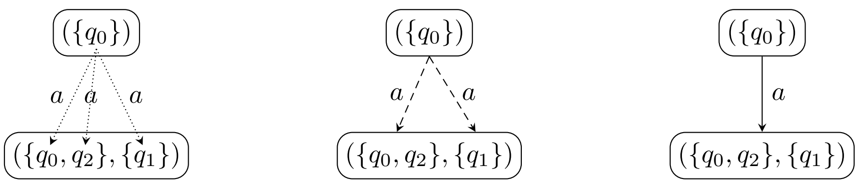

Figure 1.4: Different notions of runs.

finally allow to decide which states of the lower part of B may be accepting. In this section we will first explain the mechanical construction of B and give the intuition behind it afterwards.

There are three colours that sets of the lower part can have, let us call them 0, 1, and 2. The colour of a set says something about the history of the runs that reach this set. We have to clarify what we mean by run at this point. Conceptually, the subset-tuple construction unifies runs of the input automaton A. The construction conceptually includes two abstraction levels of this unification. Figure **??** illustrates this. The figure shows three copies of two states of the upper part of the last section. The leftmost pair shows in dotted lines the runs of the original automaton A. These runs go from A-state to A-state, and are the ones that are unified by the construction. The middle part shows the conceptual unification of the A-runs to at most two outgoing branches per subset-tuple state, one for the accepting successors and one for the non-accepting successors. These runs go from state set to state set and correspond to the run analysis done with reduced split trees. The rightmost part finally shows the real run of the automaton excerpt. This is the run that is seen from the outside, when the inner structure of the states is now known. It unifies all the A-runs to one single run.

In the following we will always refer to the notion of run in the middle of Figure **??**. That is, the notion that directly corresponds to reduced split trees. This conceptual view unifies and simplifies the A-runs as much as possible, but still guards enough information for figuring out a correct acceptance behaviour of the final complement automaton B.

A run arriving at a set is thus a branch of a corresponding reduced split tree. It can be obtained by starting at the node corresponding to the set in question and following the edges upwards toward the root of the tree. A given set may occur in many reduced split trees, as there is a reduced split tree for every word of A's alphabet. The set of runs arriving at a set are thus the corresponding branches of all the reduced split tree where the set occurs.

In the construction of the lower part, we are interested in the history of the runs back until

q_{pred}: no 2-coloured sets	s non-accepting	s accepting
$c(s_{pred}) = 0$	0	2
$c(s_{pred}) = 1$	2	2

q_{pred}: one or more 2-coloured sets	s non-accepting	s accepting
$c(s_{pred}) = 0$	0	1
$c(s_{pred}) = 1$	1	1
$c(s_{pred}) = 2$	2	2

Figure 1.5: Colour rules.

the time when they left the upper part. The crucial information is whether this history of a run includes a so-called right-turn. The notion of right-turn can be understood figuratively. In a reduced split tree, a run can be thought of as having at any node p the choice of either going to the accepting child of p or to the non-accepting one. Since in the right-to-left version of reduced split trees accepting children are to the right of non-accepting children, the run literally "turns right" when going to the accepting child. Consequently, if a run has a right-turn in its history, then it has visited at least one accepting set since leaving the upper part. On the other hand, if a run has no right-turns in its history, then it has visited no accepting sets since leaving the upper part.

That leads us back to the colours that we use for labelling the sets of the lower part. The meaning of the colours 0, 1, and 2 is the following.

- 2: the run includes a right-turn in the lower part
- 1: the run includes a right-turn in the lower part, but in the B-state where the run visited the accepting child, there was already another set with colour 2
- 0: the run does not include right-turns in the lower part

The role of colour 0 and colour 2 should be clear from the above explanations. The role colour 1 is more subtle and we will explain it later in this section when we give the intuition behind the selection of the accepting states of B. For now, we will complete the description of how to construct the lower part and thereby the final complement automaton B.

As mentioned, constructing the states of the lower part is done in the same way as constructing the states of the upper part, with the difference that every set s is assigned a colour. This colour depends on the colour of the predecessor set s_{pred} of s and on whether s itself is an accepting or non-accepting set. Furthermore, there are different rules for the two cases where the B-state p containing s_{pred} contains one or more 2-coloured sets or does not contain any 2-coloured sets. Figure 1.5 contains the complete rules for determining the colour of set s. Note that states of the upper part are treated as all their sets would have colour 0.

The colour rules are in fact simple. The first rows in the two matrices in Figure 1.5 treat the case where the run was still "clean" when it arrived at s's predecessor s_{pred}. If now s is the non-accepting child of s_{pred}, then the run stays clean and s gets colour 0. But if s is the accepting child, then the run just commits its first right-turn and gets dirty. Depending on whether there is another 2-coloured set in the state, s gets either colour 1 or colour 2. The remaining rows in the matrices of Figure 1.5 express the continuation of "dirtiness".

1.1 Optimisations

1.1.1 Removal of Non-Accepting States (R2C)

1.1.2 Merging of Adjacent Sets (M)

1.1.3 Reduction of 2-Coloured Sets (M2)

Bibliography

[1] S. Breuers, C. Löding, J. Olschewski. Improved Ramsey-Based Büchi Complementation. In L. Birkedal, ed., *Foundations of Software Science and Computational Structures*. vol. 7213 of *Lecture Notes in Computer Science*. pp. 150–164. Springer Berlin Heidelberg. 2012.

[2] J. R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *Proc. International Congress on Logic, Method, and Philosophy of Science, 1960*. Stanford University Press. 1962.

[3] S. J. Fogarty, O. Kupferman, T. Wilke, et al. Unifying Büchi Complementation Constructions. *Logical Methods in Computer Science*. 9(1). 2013.

[4] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley. 2nd edition ed.. 2001.

[5] D. Kähler, T. Wilke. Complementation, Disambiguation, and Determinization of Büchi Automata Unified. In L. Aceto, I. Damgård, L. Goldberg, et al, eds., *Automata, Languages and Programming*. vol. 5125 of *Lecture Notes in Computer Science*. pp. 724–735. Springer Berlin Heidelberg. 2008.

[6] J. Klein. Linear Time Logic and Deterministic Omega-Automata. *Master's thesis, Universität Bonn*. 2005.

[7] J. Klein, C. Baier. Experiments with Deterministic ω-Automata for Formulas of Linear Temporal Logic. In J. Farré, I. Litovsky, S. Schmitz, eds., *Implementation and Application of Automata*. vol. 3845 of *Lecture Notes in Computer Science*. pp. 199–212. Springer Berlin Heidelberg. 2006.

[8] R. Kurshan. Complementing Deterministic Büchi Automata in Polynomial Time. *Journal of Computer and System Sciences*. 35(1):pp. 59 – 71. 1987.

[9] C. Löding. Optimal Bounds for Transformations of ω-Automata. In C. Rangan, V. Raman, R. Ramanujam, eds., *Foundations of Software Technology and Theoretical Computer Science*. vol. 1738 of *Lecture Notes in Computer Science*. pp. 97–109. Springer Berlin Heidelberg. 1999.

[10] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*. 9(5):pp. 521 – 530. 1966.

[11] M. Michel. Complementation is more difficult with automata on infinite words. *CNET, Paris*. 15. 1988.

[12] A. Mostowski. Regular expressions for infinite trees and a standard form of automata. In A. Skowron, ed., *Computation Theory*. vol. 208 of *Lecture Notes in Computer Science*. pp. 157–168. Springer Berlin Heidelberg. 1985.

[13] D. E. Muller. Infinite Sequences and Finite Machines. In *Switching Circuit Theory and Logical Design, Proceedings of the Fourth Annual Symposium on*. pp. 3–16. Oct 1963.

[14] D. E. Muller, P. E. Schupp. Simulating Alternating Tree Automata by Nondeterministic Automata: New Results and New Proofs of the Theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science.* 141(1–2):pp. 69 – 107. 1995.

[15] F. Nießner, U. Nitsche, P. Ochsenschläger. Deterministic Omega-Regular Liveness Properties. In S. Bozapalidis, ed., *Preproceedings of the 3rd International Conference on Developments in Language Theory, DLT'97.* pp. 237–247. Citeseer. 1997.

[16] M. Rabin, D. Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development.* 3(2):pp. 114–125. April 1959.

[17] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society.* 141:pp. 1–35. July 1969.

[18] M. Roggenbach. Determinization of Büchi-Automata. In E. Grädel, W. Thomas, T. Wilke, eds., *Automata Logics, and Infinite Games.* vol. 2500 of *Lecture Notes in Computer Science.* pp. 43–60. Springer Berlin Heidelberg. 2002.

[19] S. Schewe. Büchi Complementation Made Tight. In *26th International Symposium on Theoretical Aspects of Computer Science-STACS 2009.* pp. 661–672. 2009.

[20] A. P. Sistla, M. Y. Vardi, P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science.* 49(2–3):pp. 217 – 237. 1987.

[21] R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control.* 54(1–2):pp. 121 – 141. 1982.

[22] W. Thomas. Automata on Infinite Objects. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science (Vol. B).* chap. Automata on Infinite Objects, pp. 133–191. MIT Press, Cambridge, MA, USA. 1990.

[23] U. Ultes-Nitsche. A Power-Set Construction for Reducing Büchi Automata to Non-Determinism Degree Two. *Information Processing Letters.* 101(3):pp. 107 – 111. 2007.

[24] M. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller, G. Birtwistle, eds., *Logics for Concurrency.* vol. 1043 of *Lecture Notes in Computer Science.* pp. 238–266. Springer Berlin Heidelberg. 1996.

[25] M. Y. Vardi, T. Wilke. Automata: From Logics to Algorithms. In J. Flum, E. Grädel, T. Wilke, eds., *Logic and Automata: History and Perspectives.* vol. 2 of *Texts in Logic and Games.* pp. 629–736. Amsterdam University Press. 2007.

[26] Q. Yan. Lower Bounds for Complementation of ω-Automata Via the Full Automata Technique. In M. Bugliesi, B. Preneel, V. Sassone, et al, eds., *Automata, Languages and Programming.* vol. 4052 of *Lecture Notes in Computer Science.* pp. 589–600. Springer Berlin Heidelberg. 2006.