[Tho90] W. Thomas. Automata on infinite objects. *Handbook of theoretical computer science*, pages 165–191, 1990.

[Var85] M.Y. Vardi. The timing of converse: Reasoning about two-way computations. In *Logic of Programs Workshop*, volume 193, pages 413–424, Brooklyn, June 1985. Lecture Notes in Computer Science, Springer-Verlag.

[Var88] M.Y. Vardi. A temporal fixpoint calculus. In *Proc. 15th ACM Symp. on Principles of Programming Languages*, pages 250–259, San Diego, January 1988.

[Var95] M.Y. Vardi. On the complexity of modular model checking. In *Proceedings of the 10th IEEE Symposium on Logic in Computer Science*, June 1995.

[VL93] B. Vergauwen and J. Lewi. A linear local model checking algorithm for CTL. In *Proc. 4th Conferance on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 447–461, Hildesheim, August 1993. Springer-Verlag.

[VS85] M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th ACM Symp. on Theory of Computing*, pages 240–251, 1985.

[VW84] M.Y. Vardi and P. Wolper. Yet another process logic. In *Logics of Programs*, volume 164, pages 501–512. Lecture Notes in Computer Science, Springer-Verlag, 1984.

[VW86a] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

[VW86b] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–21, April 1986.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.

[Wol89] P. Wolper. On the relation of programs and computations to models of temporal logic. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proc. Temporal Logic in Specification*, volume 398, pages 75–123. Lecture Notes in Computer Science, Springer-Verlag, 1989.

[ZC93] A. Zanardo and J. Carno. An ockhamist computational logic: past-sensitive necessitation in CTL*. *Journal of Logic and Computation*, 3(3):294–268, June 1993.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symposium on Foundation of Computer Science*, pages 46–57, 1977.

[Pnu85] A. Pnueli. In transition from global to modular temporal reasoning about programs. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 123–144. Springer-Verlag, 1985.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the Sixteenth ACM Symposium on Principles of Programming Languages*, Austin, Januery 1989.

[Pra76] V.R. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proc. 17th IEEE Symposium on Foundation of Computer Science*, pages 109–121, 1976.

[Pri57] A. Prior. *Time and Modality*. Oxford University Press, 1957.

[PW84] S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations. In *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, pages 28–37, Vancouver, August 1984.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Int'l Symp. on Programming*, volume 137, pages 337–351. Springer-Verlag, Lecture Notes in Computer Science, 1981.

[Rab69] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

[Rab70] M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pages 1–23. North Holland, 1970.

[Saf89] S. Safra. *Complexity of automata on infinite objects*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1989.

[Sav70] W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. on Computer and System Sciences*, 4:177–192, 1970.

[SC85] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *J. ACM*, 32:733–749, 1985.

[SE84] R. S. Street and E. A. Emerson. An elementary decision procedure for the mu-calculus. In *Proc. 11th Int. Colloquium on Automata, Languages and Programming*, volume 172. Lecture Notes in Computer Science, Springer-Verlag, July 1984.

[Sis83] A.P. Sistla. *Theoretical issues in the design of distributed and concurrent systems*. PhD thesis, Harward University, Cambridge, MA, 1983.

[SOR93] R.E. Shankar, S. Owre, and J.M. Rushby. The PVS proof checker: A reference manual (beta release). Technical report, Computer Science laboratory, SRI International, Menlo Park, California, March 1993.

[Sti87] C. Stirling. Comparing linear and branching time temporal logics. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398, pages 1–20. Lecture Notes in Computer Science, Springer-Verlag, 1987.

[Str81] R.S. Street. Propositional dynamic logic of looping and converse. In *Proceedings of the 13th ACM Symposium on Theory of Computing*, pages 375–383, Milwaukee, 1981.

[SVW87] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[Tar72] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KP95] O. Kupferman and A. Pnueli. Once and for all. In *Proc. 10th IEEE Symposium on Logic in Computer Science*, San Diego, June 1995.

[Kup95] O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. In *Computer Aided Verification, Proc. 7th Int. Workshop*, Liege, July 1995.

[KV95] O. Kupferman and M.Y. Vardi. On the complexity of branching modular model checking. In *Proc. 6th Conferance on Concurrency Theory*, Philadelphia, August 1995.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, pages 174–185, January 1980.

[Lam83] L. Lamport. Specifying concurrent program modules. *ACM Trans. on Programming Languages and Systenms*, 5:190–222, 1983.

[Lon93] D.E. Long. *Model checking, abstraction and compositional verification*. PhD thesis, Carnegie-Mellon University, Pittsburgh, 1993.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, Brooklyn, June 1985. Springer-Verlag.

[Lyn77] N. Lynch. Log space recognition and translation of parenthesis languages. *J. ACM*, 24:583–590, 1977.

[MAB+94] Z. Manna, A. Anuchitanukul, N. Bjorner, A. Browne, E. Chang, M. Colon, L. De Alfaro, H. Devarajan, H. Sipma, and T. Uribe. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Dept. of Computer Science, Stanford University, 1994.

[McM93] K.L. McMillan. *Symbolic model checking*. Kluwer Academic Publishers, 1993.

[McN66] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489, September 1971.

[MP94] M. Mihail and C.H. Papademitriou. On the random walk method for protocol testing. In *Proc. 5th Workshop on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 132–141, Stanford, June 1994. Springer-Verlag.

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54,:267–276, 1987.

[MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Int. Colloquium on Automata, Languages and Programming*. Springer-Verlag, 1986.

[MSS88] D. E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science*, pages 422–427, Edinburgh, July 1988.

[GL91] O. Grumberg and D.E. Long. Model checking and modular verification. In *Proc. 2nd Conferance on Concurrency Theory*, volume 527 of *Lecture Notes in Computer Science*, pages 250–265. Springer-Verlag, 1991.

[GL94] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems*, 16(3):843–871, 1994.

[Gor85] M.J.C. Gordon. HOL: A Machine Oriented Formulation of Higher Order Logic. Technical Report 68, July 1985.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, pages 163–173, January 1980.

[HC68] G.E. Hughes and M.J. Cresswell. *An introduction to modal logic*. Methuen, New York, 1968.

[HK94] J.Y. Halpern and B. Kapron. Zero-one laws for modal logic. *Annals of Pure and Applied Logic*, 69:157–193, 1994.

[HKV95] T. Henzinger, O. Kupferman, and M.Y. Vardi. An automata-theoretic approach to real-time branching logics. February 1995.

[Hoa69] C.A.R. Hoare. An axiomatic basis of computer programming. *Communications of the ACM*, 12(10):576–583, 1969.

[HT87] Th. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *Proc. 14th International Coll. on Automata, Languages, and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 269–279. Springer-Verlag, 1987.

[Imm81] N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):384–406, 1981.

[JJ89] C. Jard and T. Jeron. On-line model-checking for finite linear temporal logic specifications. In *Automatic Verification Methods for Finite State Systems, Proc. Int. Workshop, Grenoble*, volume 407, pages 189–196, Grenoble, June 1989. Lecture Notes in Computer Science, Springer-Verlag.

[Jon75] N.D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–75, 1975.

[Jos87a] B. Josko. MCTL – an extension of CTL for modular verification of concurrent systems. In *Temporal Logic in Specification, Proceedings*, volume 398 of *Lecture Notes in Computer Science*, pages 165–187, Altrincham,UK, April 1987. Springer-Verlag.

[Jos87b] B. Josko. Model checking of CTL formulae under liveness assumptions. In *Proc. 14th Colloq. on Automata, Programming, and Languages (ICALP)*, volume 267 of *Lecture Notes in Computer Science*, pages 280–289. Springer-Verlag, July 1987.

[Jos89] B. Josko. Verifying the correctness of AADL modules using model chekcing. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (Proceedings of REX Workshop)*, volume 430 of *Lecture Notes in Computer Science*, pages 386–400, Mook, The Netherlands, May/June 1989. Springer-Verlag.

[Kam68] J.A.W. Kamp. *Tense Logic and the Theory of Order*. PhD thesis, UCLA, 1968.

[Kam94] M. Kaminski. A branching time logic with past operators. *Journal of Computer and System Sciences*, 49(2):223–246, October 1994.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, White Plains, October 1988.

[EJ91] E.A. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 368–377, San Juan, October 1991.

[EJS93] E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proc. 5th Int. Workshop*, volume 697, pages 385–396, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

[EL85a] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 84–96, New Orleans, January 1985.

[EL85b] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, Hawaii, 1985.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the proposoitional mu-calculus. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 267–278, Cambridge, June 1986.

[EL87] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[Eme85] E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 79–87. Springer-Verlag, 1985.

[Eme90] E.A. Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, pages 997–1072, 1990.

[Eme94] E.A. Emerson. Automated temporal reasoning about reactive systems. In *8th BANFF Higher Order Workshop*, 1994.

[ES84] E.A. Emerson and A. P. Sistla. Deciding branching time logic. In *Proceedings of the 16th ACM Symposium on Theory of Computing*, Washington, April 1984.

[FL79] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and Systems Sciences*, 18:194–211, 1979.

[Flo67] R.W. Floyd. Assigning meaning to programs. In *Proceedings Symposium on Applied Mathematics*, volume 19, 1967.

[Fra92] N. Francez. *Verification*. International Computer Science. Addison-Weflay, 1992.

[Gab87] D. Gabbay. The declarative past and imperative future. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 407–448. Springer-Verlag, 1987.

[Gei94] D. Geist. Implementing CTL^2 in the SMV package. Private communication, 1994.

[GHR91] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. A compendium of problems complete for P. Technical report, University of Washington, 1991.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, pages 142–155, Stanford, California, June 1994. Lecture Notes in Computer Science, Springer-Verlag.

[CD88] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, pages 428–437. Lecture Notes in Computer Science, Springer-Verlag, 1988.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[CGH+95] E.M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D.E. Long, K.L. McMillan, and L.A. Ness. Verification of the futurebus+ cache coherence protocol. *Formal Methods in System Design*, 6:217–232, 1995.

[CGL93] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Decade of Concurrency – Reflections and Perspectives (Proceedings of REX School)*, Lecture Notes in Computer Science, pages 124–175. Springer-Verlag, 1993.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[Cle93] R. Cleaveland. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[CLM89] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *Proc. 4th IEEE Symposium on Logic in Computer Science*, pages 353–362, 1989.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[DC86] D.L. Dill and E.M. Clarke. Automatic verification of asynchronous circuits using temporal logic. *IEEE Proceedings*, 133-E:276–282, 1986.

[DDGJ89] W. Damm, G. Döhmen, V. Gerstner, and B. Josko. Modular verification of Petri nets: the temporal logic approach. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (Proceedings of REX Workshop)*, volume 430 of *Lecture Notes in Computer Science*, pages 180–207, Mook, The Netherlands, May/June 1989. Springer-Verlag.

[DGG93] D. Dams, O. Grumberg, and R. Gerth. Generation of reduced models for checking fragments of CTL. In *Proc. 5th Conf. on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 479–490. Springer-Verlag, June 1993.

[EH85] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.

Bibliography

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 414–425, Philadelphia, June 1990.

[ASSSV94] A. Aziz, T.R. Shiple, V. Singhal, and A.L. Sangiovanni-Vincentelli. Formula-dependent equivalence for compositional CTL model checking. In *Proc. 6th Conf. on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 324–337, Stanford, CA, June 1994. Springer-Verlag.

[BB87] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 62–74. Springer-Verlag, 1987.

[BBG+94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th Workshop on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 182–193, Stanford, June 1994.

[BCD85] M. Browne, E.M. Clarke, and D.L. Dill. Automatic circuit verification using temporal logic: Two new examples. In *IEEE Int. Conf. on Computer Design: VLSI and Computers*, Port Chester, October 1985.

[BCDM86] M.C. Browne, E.M. Clarke, D.L. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Trans. on Computers*, C-35:1035–1044, 1986.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[Bee80] C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. on Database Systems*, 5:241–259, 1980.

[BG93] O. Bernholtz and O. Grumberg. Branching time temporal logic and $\mathcal{A}$mo$r\mathrm{P}\mathcal{H}$ous tree automata. In *Proc. 4th Conferance on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 262–277, Hildesheim, August 1993. Springer-Verlag.

[BG94] O. Bernholtz and O. Grumberg. Buy one, get one free !!! In *Proceedings of the First International Conference on Temporal Logic*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 210–224, Bonn, July 1994. Springer-Verlag.

[BM83] R.S. Boyer and J.S. Moore. Proof-checking, theorem-proving and program verification. Technical Report 35, Institute for Computing Science and Computer Applications, University of Texas at Austin, January 1983.

[Bro86] M.C. Browne. An improved algorithm for the automatic verification of finite state systems using temporal logic. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 260–266, Cambridge, June 1986.

we come to know these factors. Optimism, however, is permissible. After all, the worst-case bounds rarely occur in practice, and life, fortunately, are full of heuristics.

provide a PSPACE procedure for CTL* model checking of concurrent programs, and provide an explanation why this bound can not be achieved for μ-calculus model checking, and even for its alternation-free fragment. We saw that the improved bounds hold also for the fair model-checking problem. We extended the automata-theoretic framework to handle modular branching model checking as well.

Our complexity results suggest an additional view on the long-standing argument of linear versus branching temporal logics, and not necessarily in favor of the branching paradigm. Traditionally, model checking has been considered easier for the branching paradigm [EH86]. The linear time-complexity of CTL model checking compensated on its lack of expressive power and gave LTL a hard time. Yet, examining the space complexities of CTL and LTL model checking, the linear paradigm seemed to be preferable: LTL model checking for concurrent programs has been known to be in PSPACE [LP85, VW86a]. To this, the branching paradigm seemed to have no answer: the known algorithms for CTL model checking worked in a bottom-up manner and had to construct an exponential extension of the concurrent program. So, our PSPACE new bound for branching-time model checking is definitely good news for the branching paradigm. There are, however, bad news too. The advantage that CTL enjoys over LTL when time-complexity is considered expires not only when space complexity is considered, but also, as we saw here, when the complexity of modular model checking is considered. Indeed, for both logics, the modular model checking problem is in PSPACE. Moreover, the lower bound for the branching paradigm holds even when both the assumption and the guarantee are in $\forall$CTL, the universal fragment of CTL.

Then, we turned to examine branching temporal logics that collaborate with the linear paradigm. This was explicit for the logics CTL^i. There, we allowed a "bounded linearity" between the branching operators of the logics. The collaboration was implicit for the two other augmentations of branching temporal logics. When interpreted over computation trees, both existential quantification over atomic propositions and past modalities induce temporal logics in which future is branching and past is linear. We saw that the resulted logics are very expressive and that they correspond to the natural way in which designers and specifiers regard time. We saw that while the increase in the expressive power does not harm the complexity of the satisfiability problem for the augmented logics, it does increase the complexity of their model-checking problem.

The model-checking complexity of the augmented logics, as well as the improved space complexity of CTL and CTL* model-checking is only one factor, among many others, when we come to examine the practical contribution of this work. Crucial additional factors are the performances of the automata-theoretic framework and the augmented logics when applied to the verification of real-life designs and systems. Another factor is how they combine with existing tools and methods that are already in use. Some good years will probably pass before

Chapter 5

Discussion

We have examined theoretical and practical aspects of branching-time model checking.

First, we saw that alternating tree automata are the key to a comprehensive automata-theoretic framework for branching temporal logics. Not only, as was shown by Muller et al., can they be used to obtain optimal satisfiability procedures, but, as we saw here, they also make it possible to derive optimal model-checking algorithms. For linear temporal logics, the complexities of the satisfiability problem and the model-checking problem coincide. As we saw, this makes the automata-theoretic framework for these logics simple. For branching temporal logics, there is a gap between the complexities of these problems, and alternation is required in order to overcome this gap. Using alternating tree automata, satisfiability is reduced to the usual nonemptiness problem, while model checking is reduced to the 1-letter nonemptiness problem, which is much easier.

We believe that alternation, and the two different versions of the nonemptiness problem that it induces, can be used for solving additional "gap problems". The gap between the satisfiability problem and the model-checking problem becomes more significant when we turn to consider *real-time branching logics*. Real-time branching logics enable a qualitative reference to time. For example, the real-time branching logic TCTL augments the branching temporal logic CTL with clock variables. While the satisfiability problem for TCTL is undecidable, its model-checking problem is in PSPACE [ACD90]. Can alternating tree automata provide a framework for real-time branching logics and explain also this gap? We believe they can. In the case of real-time branching logics, while satisfiability is reduced to the nonemptiness problem of alternating timed tree automata, which is undecidable, we conjecture that model checking can be reduced to the 1-letter nonemptiness of alternating timed tree automata with a bounded number of clocks, which is in PSPACE [HKV95].

Using the automata-theoretic framework we were able to improved the space complexity of branching-time model-checking. In particular, we saw that alternating tree automata

Proof: Hardness in EXPTIME follows from hardness of the satisfiability problem for CTL. To prove membership in EXPTIME we extend the tableau method for CTL used in [EH85]. The puzzle in this method lays in the fact that the quotient construction does not preserve modelhood. Let K' denote the quotient structure of a Kripke structure K. A formula of the form $A\varphi_1 U \varphi_2$ which is satisfied in K might not be satisfied in K' due to cycles introduced into it. Emerson and Halpern solve this puzzle by showing that if K is a model for a CTL formula φ, then K' is "almost" model for φ (a pseudo-Hintikka model), and that a model of size exponential in the size of φ can be constructed from it. This establishes a small model property for CTL and yields a tableau-based satisfiability procedure.

The fact that CTL_{bp} assumes a finite past makes the extension of the above described method easy. The crucial point is that when past is finite, the quotient construction preserves modelhood with respect to all past-time modalities. Indeed, paths that get stuck in a cycle introduced into K' do not reach the initial state and can thus be ignored. Hence, if K is a model for a CTL_{bp} formula, then K' is a pseudo-Hintikka model for it, in which the only eventualities that may not be fulfilled are of the form AU. We now sketch how, given K', we construct an exponential size model for φ from it. As a first step we construct a Kripke structure, K'', in which every state satisfies all its subformulas that have an outermost future modality (modulo the assumption that every state satisfies its subformulas that have an outermost past modality). This is done using the "matrix construction" for CTL [EH85]. Since each transition in K'' exists in K', then each state in K'' satisfies also all its subformulas that have an outermost and universal past modality. It is thus left to worry only about subformulas that have an outermost existential past modality. Some of these subformulas may be satisfied in K'' but for some it may be required to add transitions to K'. Still, these transitions are contained in the transitions of K' and can be added to K'' preserving the fulfillment of AU formulas. □

Theorem 4.24 *The satisfiability problem for CTL_{lp} is EXPTIME-complete.*

Proof: Hardness in EXPTIME follows from hardness of the satisfiability problem for CTL. Membership in EXPTIME follows from the linear translation of CTL_{lp} formulas into EQCTL_t formulas and Theorem 4.10. □

Theorem 4.22 *The model-checking problem for CTL_{lp} is PSPACE-hard.*

Proof: We prove hardness in PSPACE using the same reduction used in [SC85] for proving that model checking for LTL is PSPACE-hard. There, Sistla and Clarke associate with a polynomial space Turing machine M and an input word w, a Kripke structure K and an LTL formula ψ, such that $K \models \psi$ iff M accepts w. The formula ψ uses the X operator to describe the possible successors of a configuration of M and uses the F operator to ensure that an accepting configuration is eventually reached. A similar formula, that uses the operators F and Y can be written in CTL_{lp}. The formula is of the form $EF\xi$, where ξ is a past LTL_p formula, asserting that the current configuration is accepting, and that it has been reached by a valid run of M on w. As ψ, the length of ξ is polynomial in M and w. □

So, while the model-checking problem for CTL_{bp} is in linear time, as is the one for CTL [CES86], augmenting CTL with linear past makes its model-checking problem PSPACE-hard. Typically, the gap follows from the fact that CTL_{lp} actually subsumes the expressive power of linear temporal logic. Since the linear-past (rather than the branching-past) approach corresponds to the natural way branching temporal logics have been used to represent computations, these news are, all in all, sad. The problem of finding a tight bound for CTL_{lp} model checking is still open. While an EXPTIME upper bound is straightforward (e.g., by the linear translation to EQCTL), we did not find an EXPTIME lower bound. A good excuse for this is our conjecture that the problem is PSPACE-complete. In more details, the automata-theoretic framework from Chapter 3 can be used to reduce the model-checking problem for CTL_{lp} to the 1-letter nonemptiness problem of weak alternating automata. Unlike CTL, where the automata are of linear size, the automata for CTL_{lp} are of exponential size. We believe that, as with CTL, the structure of these automata enables a space-efficient 1-letter nonemptiness procedure.

4.3.4 Satisfiability Complexity

As with model checking, there are two interpretations of the satisfiability problem for a branching temporal logic which is sensitive to unwinding. The first interpretation, which is the one appropriate to branching past, asks whether there exists a Kripke structure K and a state w^0 in it, such that w^0 has no predecessors and $w^0 \models \varphi$. In the second interpretation, which is the one appropriate to linear past, we are given φ and are asked to determine whether there exists a computation tree $\langle T, V\rangle$ such that $\langle T, V\rangle \models \varphi$. In this section we consider satisfiability complexity for the two interpretations.

Theorem 4.23 *The satisfiability problem for CTL_{bp} is EXPTIME-complete.*

4.3.3 Model-Checking Complexity

As in Section 4.2.2, we distinguish between two interpretation to the model-checking problem The first interpretation, which is the one appropriate to branching past, asks whether $w^0 \models \varphi$. In the second interpretation, which is the one appropriate to linear past, we are given φ and K and are asked to determine whether $\langle T_K, V_K \rangle \models \varphi$. In this section we consider model-checking complexity for the two interpretations.

Theorem 4.20 *The model-checking problem for CTL_{bp} is in linear time.*

Proof: We present a model-checking procedure for CTL_{bp}. Our procedure is a simple extension of the efficient model-checking procedure for CTL in [CES86], and is of complexity linear in both the length of the formula and the size of the Kripke structure being checked. As there, the algorithm labels with a formula φ exactly all the states that satisfy φ. This is done by recursively labeling the Kripke structure with the subformulas of φ. Once the Kripke structure is labeled with the subformulas of φ, it is possible to label it also with φ. Handling of past-time modalities is symmetric to the one suggested in [CES86] for future-time modalities, switching successors and predecessors. Careful attention, however, should be payed to the fact that past is finite. While finiteness of the past does not influence checking of $E\varphi_1 S \varphi_2$, it does influence checking of $E\varphi_1 \tilde{S} \varphi_2$. When past is finite, a state w for which there exists a path from w^0 to w such that all the states in this path satisfy φ_2, satisfies $E\varphi_1 \tilde{S} \varphi_2$. Accordingly, labeling w^0 with a fresh atomic proposition $init$, we have that $E\varphi_1 \tilde{S} \varphi_2 \sim E\varphi_2 S(\varphi_2 \wedge (init \vee \varphi_1))$. Thus, the modality $E\tilde{S}$ is handled using the same procedure that handles the modality ES. □

Theorem 4.21 *The model-checking problem for $CTL^{\star}_{bp}$ is PSPACE-complete.*

Proof: Hardness in PSPACE follows from hardness of the model-checking problem for CTL*. To prove membership in PSPACE, we present a PSPACE model-checking algorithm. Our algorithm uses the PSPACE model-checking algorithm for LTL_p [Var88] and it is based on the method of reducing branching-time model checking to linear-time model checking [EL85a]. According to this method, nested formulas of the form $E\xi$ are evaluated by recursive descent. For example, in order to model check $EXEXGp$, we first model check $EXGp$ using the model checker for LTL and label every state that satisfies it with a fresh atomic proposition q. Then, we model check EXq. In order to adopt this method for $\mathrm{CTL}^{\star}_{bp}$, we should guarantee that the model checker for LTL_p considers only paths that start in the initial state. This can be easily done by labeling the initial state with a fresh atomic proposition $init$ and conjuncting each linear-time formula checked with $Pinit$. For example, in order to check $EXEXPp$, we first model check, in PSPACE, the formula $E((XPp) \wedge (Pinit))$ and label every state that satisfies it with a fresh atomic proposition q. Then we model check, again in PSPACE, the formula $E((Xq) \wedge (Pinit))$. It is easy to see that the overall complexity is PSPACE. □

- $spread(E\varphi_1 S\varphi_2, p) = AG(\varphi_2 \rightarrow p) \wedge AG(p \rightarrow AX((\varphi_1 \vee \varphi_2) \rightarrow p))$.
- $spread(E\varphi_1 \tilde{S}\varphi_2, p) = (p \leftrightarrow \varphi_2) \wedge AG(p \rightarrow AX(\varphi_2 \rightarrow p)) \wedge AG((\varphi_2 \wedge \varphi_1) \rightarrow p)$.

The definition of $spread(\varphi, p)$ guarantees that if a Kripke structure K satisfies $spread(\varphi, p)$, then for every state w for which $w \models \varphi$, we have $w \models p$. We now define the formula $label(\varphi, p)$ which guarantees that the labeling of p is tight.

$$label(\varphi, p) = spread(\varphi, p) \wedge \forall r(spread(\varphi, r) \rightarrow AG(p \rightarrow r)).$$

If a Kripke structure K satisfies $label(\varphi, p)$, then for every state w, we have $w \models p$ iff $w \models \varphi$. Once $label(\varphi, p)$ is defined, we proceed as in the linear-past case. As there, the translation is linear. Note that the fact that past is finite plays a crucial role in our translation. Only thanks to it we are able to determine labeling of $E\varphi_1 \tilde{S}\varphi_2$ in w^0 and then to spread labeling forward. To see this, consider for example the formula $\varphi = E\boldsymbol{false}\tilde{S}q$ ("exists a path in which q holds always in the past"). The first conjunct in $spread(\varphi, p)$ is $p \leftrightarrow q$ and it determines the labeling in w^0. If we eliminate it (as we should have done if past was not finite), we get

$$label(\varphi, p) = AG(p \rightarrow AX(q \rightarrow p)) \wedge \forall r[(AG(r \rightarrow AX(q \rightarrow r)) \rightarrow AG(p \rightarrow r))].$$

Thus, we get that $label(\varphi, p)$ is equivalent to $AG\neg p$ and no state, in no Kripke structure, is labeled as satisfying φ. □

As QCTL satisfies the finite model property, Theorem 4.18 implies that CTL_{bp} satisfies the finite model property as well. The logic POTL, which essentially differs from CTL_{bp} in allowing infinite past, does not satisfy this property [PW84]. Indeed, the fact that CTL_{bp} assumes a finite past, makes it an "easy" language. On the other hand, CTL_{bp} does not satisfy the tree model property. To see this, consider the formula $EF((EYp) \wedge EY\neg p)$ which is satisfied in K_1, yet no computation tree can satisfy it. As EQCTL_t does satisfy the tree model property, we could not do without universal quantifiers in the translation.

Theorem 4.19 *$CTL^\star_{lp} > CTL_{lp}$, $CTL^\star_{bp} > CTL_{bp}$, and $CTL^\star_{bp} > CTL^\star$.*

Proof: In [EH86], Emerson and Halpern show that the $\text{CTL}^\star$ formula $EGFp$ has no equivalent of CTL. It is easy to extend their proof to consider also CTL_{lp} and CTL_{bp}. Hence, $\text{CTL}^\star_{lp} > \text{CTL}_{lp}$ and $\text{CTL}^\star_{bp} > \text{CTL}_{bp}$. In the proof of Theorem 4.18, we point on a CTL_{bp} formula φ which distinguishes between a Kripke structure and its unwinding. Since $\text{CTL}^\star$ is not sensitive to unwinding, φ has no equivalence of $\text{CTL}^\star$. Hence, $\text{CTL}^\star_{bp} > \text{CTL}^\star$. □

The specification "q holds at all even places" is expressible in EQCTL_{lp} using the formula $\varphi = \exists p(p \wedge AG(p \rightarrow AXAXp) \wedge AG(p \rightarrow q))$. Extending Wolper's result from [Wol83], φ has no equivalent of CTL*. Hence, as $\mathrm{CTL}_{lp} \leq$ CTL*, we have $\mathrm{CTL}_{lp} < \mathrm{EQCTL}_{lp}$.

To prove $\mathrm{EQCTL}_{lp} = \mathrm{EQCTL}_t$, we prove that $\mathrm{EQCTL}_{lp} \leq \mathrm{EQCTL}_t$. Equivalence then follows by syntactic containment. Given an EQCTL_{lp} formula ψ, we translate ψ into an equivalent EQCTL_t formula. Let φ be a formula of the form $Y\varphi_1$ or $\varphi_1 S \varphi_2$, and let p be a fresh atomic proposition. We define the formula $label(\varphi, p)$ as follows:

- $label(Y\varphi_1, p) = \neg p \wedge AG(\varphi_1 \rightarrow AXp) \wedge AG(\neg\varphi_1 \rightarrow AX\neg p)$.
- $label(\varphi_1 S \varphi_2, p) = (p \leftrightarrow \varphi_2) \wedge AG(p \rightarrow AX(p \leftrightarrow (\varphi_1 \vee \varphi_2))) \wedge AG(\neg p \rightarrow AX(p \leftrightarrow \varphi_2))$.

The definition of $label(\varphi, p)$ guarantees that if a computation tree $\langle T, V\rangle$ satisfies $label(\varphi, p)$, then for every node $x \in T$, we have $x \models p$ iff for every path $\rho \subseteq T$ and $j \geq 0$ with $\rho_j = x$, we have $\rho, j \models \varphi$. The above observation is the key to our translation. Given ψ, we translate it into an equivalent EQCTL_t formula by replacing its path subformulas φ of the form $Y\varphi_1$ or $\varphi_1 S \varphi_2$, with a fresh atomic proposition p_φ, conjuncting the resulted formula with $label(\varphi, p_\varphi)$, and prefixing it with $\exists p_\varphi$. Replacement continues for the past formulas in $label(\varphi, p_\varphi)$, if exist. It is easy to see that the translation is linear. For example, the formula $AXAF(p \wedge Yp)$ is translated to the formula $\exists q AXAF(p \wedge q) \wedge \neg q \wedge AG(p \rightarrow AXq) \wedge AG(\neg p \rightarrow AX\neg q)$. □

Theorem 4.18 *$CTL < CTL_{bp} < QCTL_{bp} = QCTL$.*

Proof: Consider the CTL_{bp} formula $\varphi = EF((EYp) \wedge EY\neg p)$ and consider the Kripke structures K_1 and K_2 presented in Figure 4.5. It is easy to see that $K_1 \models \varphi$ and $K_2 \not\models \varphi$. As K_2 can be obtained by unwinding K_1 and as CTL is not sensitive to unwinding, no CTL formula can distinguish between K_1 and K_2. Hence CTL $<$ CTL_{bp}. As with linear past, $\mathrm{CTL}_{bp} < \mathrm{QCTL}_{bp}$ follows from the inexpressibility of "q holds at all even places" in CTL_{bp}.

To prove $\mathrm{QCTL}_{bp} =$ QCTL, we suggest a translation of QCTL_{bp} formulas into QCTL formulas. We assume a normal form for QCTL_{bp} in which the allowed past operators are EY, ES, and $E\tilde{S}$. Intuitively, we would have liked to do something similar to the translation of CTL_{lp} formulas into EQCTL. However, since a state in a Kripke structure may have several predecessors, which do not necessarily agree on the formulas true in them, we cannot do it. Instead, we label K in two steps: for every past formula φ, we first label with p_φ all the states that satisfy φ. Then we require p_φ to be the least such labeling, guaranteeing that *only* states that satisfy φ are labeled. Let φ be a formula of the form $EY\varphi_1$, $E\varphi_1 S\varphi_2$, or $E\varphi_1 \tilde{S}\varphi_2$, and let p be a fresh atomic proposition. We define the formula $spread(\varphi, p)$ as follows:

- $spread(EY\varphi_1, p) = AG(\varphi_1 \rightarrow AXp)$.

Lemma 4.15 *Let $E\psi$ be a $CTL^\star_{lp}$ formula all of whose state subformulas are in $CTL^\star$. $E\psi$ is congruent to a disjunction of formulas of the form $p \wedge Eq$ where p is a past LTL_p formula and Eq is a $CTL^\star$ formula.*

Proof: By the Separation Theorem, ψ is congruent to a boolean combination, ψ', of future and past LTL_p formulas. Without loss of generality, ψ' is of the form $\bigvee_{1 \le i \le n}(p_i \wedge q_i)$, where for all $1 \le i \le n$, p_i is a past LTL_p formula and q_i is a future LTL_p formula. As past is linear, path quantification over past LTL_p formulas can be eliminated using the congruences below.

$$E \bigvee_{1 \le i \le n} (p_i \wedge q_i) \approx \bigvee_{1 \le i \le n} E(p_i \wedge q_i) \approx \bigvee_{1 \le i \le n} (p_i \wedge Eq_i).$$

□

Theorem 4.16 $CTL^\star_{lp} = CTL^\star$.

Proof: Given a $\mathrm{CTL}^\star_{lp}$ formula φ, we translate φ into an equivalent $\mathrm{CTL}^\star$ formula. The translation proceeds from the innermost state subformulas of φ, using Lemma 4.15 to propagate past outward. Formally, we define the depth of a state subformula ξ in φ as the number of nested E's in ξ, and proceed by induction over this depth. Subformulas of depth 1 have atomic propositions as their subformulas and therefore they satisfy the lemma's condition. Also, at the end of step i of the induction, all subformulas of depth i are written as disjunctions of formulas of the form $p \wedge Eq$ where p is a past LTL_p formula and Eq is a $\mathrm{CTL}^\star$ formula. Thus, propagation can continue. In particular, at the end of the inductive propagation, φ is written as such a disjunction. Then, as the past formulas refer to the initial state, we replace Yq with ***false***, replace pSq with q, and we end up with a $\mathrm{CTL}^\star$ formula. □

As our semantics allows past to go beyond the present, Theorem 4.16 is much stronger than the $\mathrm{PCTL}^\star = \mathrm{CTL}^\star$ result in [HT87]. In all the $L_1 < L_2$ relations in Theorems 4.17, 4.18, and 4.19 below, the $L_1 \le L_2$ part follows by syntactic containment and we prove only strictness.

Theorem 4.17 $CTL < CTL_{lp} < EQCTL_{lp} = EQCTL_t$.

Proof: In [EH86], Emerson and Halpern show that the $\mathrm{CTL}^\star$ formula $AF(p \wedge Xp)$ has no equivalent of CTL. We prove that the CTL_{lp} formula $AXAF(p \wedge Yp)$ is equivalent to $AF(p \wedge Xp)$, thus it has no equivalent of CTL. Consider a computation tree $\langle T, V\rangle$. Clearly, $\langle T, V\rangle \models AF(p \wedge Xp)$ iff for every path $\rho \subseteq T$, there exists $j \ge 0$ such that $\rho, j \models p \wedge Xp$, or, equivalently, there exists $i \ge 1$ such that $\rho, i \models p \wedge Yp$. This holds iff for every path $\rho \subseteq T$, $\rho, 1 \models F(p \wedge Yp)$, which holds iff $\langle T, V\rangle \models AXAF(p \wedge Yp)$. Hence, CTL $<$ CTL_{lp}.

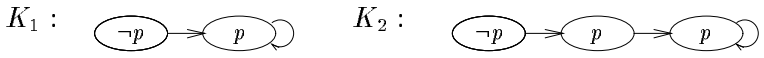


Figure 4.5: The Kripke structures K_1 and K_2.

Proof: Consider the Kripke structure K_1 appearing in Figure 4.5. The computation tree induced by K_1 is $\langle T_{K_1}, V_{K_1} \rangle$ where $T_{K_1} = 0^*$ and V_{K_1} is defined by $V_{K_1}(\epsilon) = \emptyset$ and $V_{K_1}(x) = \{p\}$ for all $x \in 0^+$. It is easy to see that while $K_1 \not\models AF(p \wedge AYp)$, we have $\langle T_{K_1}, V_{K_1} \rangle \models AF(p \wedge AYp)$. □

Note that since $\mathrm{CTL}^\star_{bp}$ assumes a finite past, both K_1 and $\langle T_{K_1}, V_{K_1} \rangle$ satisfy $AGAP\neg p$. Also, as both w^0 and ϵ have no predecessors, then $w^0 \not\models EYt$ and $\epsilon \not\models EYt$. Clearly, for all $w \neq w^0$ and $x \neq \epsilon$, we have $w \models EYt$ and $x \models EYt$. Thus, both $\mathrm{CTL}^\star_{bp}$ and $\mathrm{CTL}^\star_{lp}$ can characterize the starting point of the past.

Expressive power

In this section we consider the expressive power of branching temporal logics with past with respect to branching temporal logics without past. Our results are summarized in the hierarchy presented in Figure 4.6. In the Figure, we use $L_1 \leftarrow L_2$ to indicate that $L_1 > L_2$, we use $L_1 \leftrightarrow L_2$ to indicate that $L_1 = L_2$, and we use $L_1 \text{ — } L_2$ to indicate that $L_1 \not\geq L_2$ and $L_2 \not\geq L_1$.

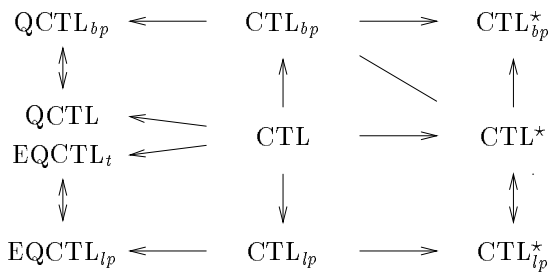


Figure 4.6: Hierarchy of expressive power.

We first prove that $\mathrm{CTL}^\star_{lp} = \mathrm{CTL}^\star$. While $\mathrm{CTL}^\star_{lp}$ is interpreted over computation trees, $\mathrm{CTL}^\star$ is interpreted over Kripke structures. However, as $\mathrm{CTL}^\star$ is insensitive to unwinding, this causes no difficulty. We use the Separation Theorem for LTL_p, quoted below.

Theorem 4.14 *[Gab87] Any LTL_p formula is congruent to a boolean combination of past and future LTL_p formulas.*

We consider also the logic QCTL_{bp}, obtained by adding both universal and existential quantification over atomic propositions to CTL_{bp}. Precisely, every CTL_{bp} formula is a QCTL_{bp} formula and, in addition, if ψ is a QCTL_{bp} formula and p is an atomic proposition occurring free in ψ, then $\exists p\psi$ is also a QCTL_{bp} formula. The semantics of $\exists p\psi$ is given by $K \models \exists p\psi$ iff there exists a Kripke structure K' such that $K' \models \psi$ and K' differs from K in at most the labels of p. We use $\forall p\psi$ to abbreviate $\neg\exists p\neg\psi$.

We now turn to the logic $\mathrm{CTL}^{\star}_{lp}$. We define its semantic with respect to a computation tree $\langle T, V\rangle$. We use $x \models \varphi$ to indicate that a state formula φ holds at node $x \in T$. We use $\rho, j \models \psi$ to indicate that a path formula ψ holds in position j of the the path $\rho \subseteq T$. The relation $\models$ is defined similarly to the one of $\mathrm{CTL}^{\star}_{bp}$, taking a node x here instead a state w there, and a path ρ here, instead π there. In particular, we have:

- $x \models E\psi_1$ iff there exist a path ρ and $j \geq 0$ such that $\rho_j = x$ and $\rho, j \models \psi_1$.
- $\rho, j \models \varphi$ for a state formula φ, iff $\rho_j \models \varphi$.

For a computation tree $\langle T, V\rangle$, we say that $\langle T, V\rangle \models \varphi$ iff $\epsilon \models \varphi$.

The logic LTL_p is an extension of the linear temporal logic LTL. It extends LTL by allowing also the past-time operators Y and S. Recall that the logic CTL_{lp} is the linear-past extension of CTL. As past is linear, path quantification of past-time operators is redundant. Thus, in CTL_{lp} we require the temporal operators X and U be to preceded by a path quantifier, yet we impose no equivalent restriction on the operators Y and S. This implies that in CTL_{lp}, path quantifiers are followed by LTL_p formulas that have in them a single, and outermost, future-time operator. We consider also the logic EQCTL_{lp}, obtained by adding existential quantification over atomic propositions to CTL_{lp}. The logic EQCTL_{lp} is interpreted over computation trees, as EQCTL_{t}.

4.3.2 Expressiveness

Branching past versus linear past

Unwinding of a Kripke structure has two implications on past: it makes past linear and it makes past finite. We have already pointed on the strong relation between quantification over atomic propositions and past-time operators. In some sense, that we clarify later, quantified atomic propositions can be viewed as *history variables* that record events that happened in the past. It is not surprising then that once we augment a branching temporal logic with past-time operators, it becomes sensitive to unwinding. Hence, the two approaches we present for past induce different logics.

Theorem 4.13 *Satisfaction of $CTL^{\star}_{bp}$ is sensitive to unwinding.*

- If ψ_1 and ψ_2 are $\mathrm{CTL}^\star_{bp}$ (or $\mathrm{CTL}^\star_{lp}$) path formulas, then so are $Y\psi_1$, and $\psi_1 S \psi_2$.

We consider also the logics CTL_{bp} and CTL_{lp} which augment CTL with the temporal operators Y and S. We use the following abbreviations in writing formulas:

- $P\psi = tS\psi$ ("sometime in the past").
- $H\psi = \neg P \neg \psi$ ("always in the past").
- $\psi_1 \tilde{S} \psi_2 = \neg((\neg\psi_1) S (\neg\psi_2))$ (duality for the Since operator).

A *past formula* is a formula in which no future-time operators occur. Similarly, a *future formula* is a formula in which no past-time operators occur.

The difference between $\mathrm{CTL}^\star_{bp}$ and $\mathrm{CTL}^\star_{lp}$ arises when we consider their semantics. Let us starts with $\mathrm{CTL}^\star_{bp}$. As $\mathrm{CTL}^\star$, we define the semantics of $\mathrm{CTL}^\star_{bp}$ with respect to a Kripke structure $K = \langle AP, W, R, w^0, L\rangle$. Defining the semantics of $\mathrm{CTL}^\star$, satisfaction of a path formula at some position in the path is independent of earlier positions in the path. This is not the case with $\mathrm{CTL}^\star_{bp}$, where a reference to past is possible. In $\mathrm{CTL}^\star_{bp}$, we define the semantics of path formulas with respect to both a path and a position in it. As with $\mathrm{CTL}^\star$, we use $w \models \varphi$ to indicate that a state formula φ holds at state w. We use $\pi, j \models \psi$ to indicate that a path formula ψ holds at position j of the path π. For $\varphi = \boldsymbol{true}$, $\boldsymbol{false}$, $p \in AP$, $\neg\varphi$, or $\varphi_1 \vee \varphi_2$, the relation $w \models \varphi$ is identical to the one in $\mathrm{CTL}^\star$. Below we give the semantics for $\varphi = E\psi_1$ and for path formulas.

- $w \models E\psi_1$ iff there exist a path $\pi = w_0, w_1, \ldots$ and $j \geq 0$ such that $w_j = w$ and $\pi, j \models \psi_1$.
- $\pi, j \models \varphi$ for a state formula φ, iff $w_j \models \varphi$.
- $\pi, j \models \neg\psi_1$ iff $\pi, j \not\models \psi_1$.
- $\pi, j \models \psi_1 \vee \psi_2$ iff $\pi, j \models \psi_1$ or $\pi, j \models \psi_2$.
- $\pi, j \models X\psi_1$ iff $\pi, j+1 \models \psi_1$.
- $\pi, j \models Y\psi_1$ iff $j > 0$ and $\pi, j-1 \models \psi_1$.
- $\pi, j \models \psi_1 U \psi_2$ iff there exists $k \geq j$ such that $\pi, k \models \psi_2$ and $\pi, i \models \psi_1$ for all $j \leq i < k$.
- $\pi, j \models \psi_1 S \psi_2$ iff there exists $0 \leq k \leq j$ such that $\pi, k \models \psi_2$ and $\pi, i \models \psi_1$ for all $k < i \leq j$.

Note that the past-time operator Y is interpreted in the strong sense. That is, in order to satisfy a Y requirement, a state must have some predecessor.

is not more expressive than CTL*. Another augmentation of CTL* with past-time modalities is the *ockhamist computational logic* (OCL), presented in [ZC93]. We found the semantics of OCL unsatisfactory, as it is interpreted over structures which are not fusion closed.

In this section, based on [KP95], we consider the logics CTL$^\star_{bp}$ and CTL$^\star_{lp}$, as well as their sub-languages CTL$_{bp}$ and CTL$_{lp}$. Syntactically, CTL$^\star_{bp}$ and CTL$^\star_{lp}$ are exactly the same: both extend the full branching temporal logic CTL* with past-time modalities. Semantically, we have two completely different interpretations. Formulas of CTL$^\star_{bp}$ are interpreted over states of a Kripke structure with a definite initial state. Since each state in a Kripke structure may have several successors and predecessors, this interpretation induces a "branching reference" to both future and past. Accordingly, we regard CTL$^\star_{bp}$ formulas as we did with formulas of EQCTL*. That is, a formula of CTL$^\star_{bp}$ describes a single computation which is a partially ordered set [PW84]. The initial state corresponds to a single initial process, a state with several successors corresponds to creating new processes, and a state with several predecessors corresponds to merging processes. Unlike [PW84], we consider only paths that start in the initial state and thus ignore computations which can be traversed backwards an infinite number of steps. For example, the CTL$^\star_{bp}$ formula $AG(term \rightarrow ((EPterm_1) \wedge \ldots \wedge (EPterm_n)))$ states that the entire system can terminate only after all its n processes have terminated. The semantics of CTL$^\star_{lp}$, on the other hand, correspond to the semantics of EQCTL$^\star_t$. That is, formulas of CTL$^\star_{lp}$ are interpreted over nodes of a computation tree obtained by, say, unwinding a Kripke structure. Since each node in a computation tree may have several successors but only one predecessor (except the root that has no predecessor), this interpretation induces a linear reference to past and a branching reference to future. Accordingly, we regard CTL$^\star_{lp}$ formulas as describing a set of computations of a nondeterministic program, where each computation is a totally ordered set. The branching in the tree represents non-determinism or interleaving due to concurrency. For example, the CTL$^\star_{lp}$ formula $AG(grant \rightarrow P(req))$ states that grant is given only upon request. Note that there is no path quantification over $P(req)$. It is clear from the semantics that a request should be found along the path that led from the root to the state in which the grant is received. As with EQCTL* and EQCTL$^\star_t$, the two different views to past not only induce two different practices as specification languages but also differ in their expressive power and model-checking complexities.

4.3.1 The logics CTL$^\star_{bp}$ and CTL$^\star_{lp}$

Recall that CTL$^\star_{bp}$ and CTL$^\star_{lp}$ have the same syntax, extending CTL* by allowing past-time operators. Formally, we obtain CTL$^\star_{bp}$ and CTL$^\star_{lp}$ by augmenting CTL* by the two temporal operators Y ("Yesterday") and S ("Since"). Thus, we add to the syntax of CTL* the following rule:

the formulation of specifications more intuitive and does not increase the complexity of the validity and the model-checking problems [LPZ85, Var88].

Is the same true of branching temporal logics? Examining this question, we found in the literature several logics that extend branching temporal logics with past-time modalities. Yet, as we soon specify, we did not find a logic that meets our understanding of past in a branching-time model: we distinguish between two possible views regarding the nature of past. In the first view, *past is branching* and each moment in time may have several possible futures and several possible pasts. In the second view, *past is linear* and each moment in time may have several possible futures and a unique past. Both views assume that *past is finite*. Namely, they consider only paths that start at a definite starting time. There is much similarity between the two views here and the two semantics we considered for branching temporal logics augmented with existential quantification over atomic propositions. We will get back to this point later.

Before going on with our two views, let us consider the branching temporal logics with past that we found in the literature. The original version of *propositional dynamic logic* (PDL), as presented by Pratt in [Pra76], includes a *converse construction*. The converse construction reverses the program, thus enabling the specifications to refer to the past. As each state in a program may have several predecessors, *converse-PDL* corresponds to the branching-past interpretation. Beyond our aspiration to replace the PDL system with branching temporal logics used nowadays, our main complaint about the converse construction is that it allows infinite paths in the reversed programs. Thus, it does not reflect the (helpful, as we shall show) fact that programs have a definite starting time. As a result, combining the converse construction with other constructions, e.g. the *loop* construction and the *repeat* construction, results in quite complicated logics [Str81, Var85, VW86b]: they do not satisfy the *finite model property*, their decidability becomes more expensive, and no model-checking procedures are presented for them. In addition, while *converse*-DPDL satisfies the *tree model property* [VW86b], the logics we introduce for the branching-past interpretation do not satisfy it. So, intuitively, our branching past is "more branching". In spite of this, our logics satisfy the finite model property. The same tolerance towards paths that backtrack the transition relation without ever reaching an initial state is found in *POTL*, which augments the branching temporal logic $B(X, F, G)$ with past-time modalities [PW84], in the more expressive logic EBTL [Kam94], and in the reverse operators in [Sti87].

A logic *PCTL**, which augments the branching temporal logic CTL* with past-time modalities, is introduced in [HT87]. Formulas of PCTL* are interpreted over *computation trees*. Thus, PCTL* corresponds to the linear-past interpretation. However, the semantics of PCTL* makes the usage of past-time modalities very limited. Actually, past cannot go beyond the present. For example, the PCTL* formula $EXEYtrue$ ("exists a successor state for which there exists a predecessor state that satisfies *true*") is not satisfiable. It is not surprising then, that PCTL*

alternating_path(s,t) holds iff $K_G \models_t \varphi$. Note that, as with $G2(p)$, the formula φ is not appropriate for the structure semantics. □

The fact that the program complexity of EQCTL and EQCTL* is in P is quite surprising. Mysteriously, while model checking in the tree semantics is harder than model checking in the structure semantics, we have that the program complexity of model checking is lower in the tree semantics. The elucidation of this mystery lies in the fact that the model-checking problem for EQ$\mathcal{L}_t$ is closer to the satisfiability problem for $\mathcal{L}$ than the model-checking problem for EQ$\mathcal{L}$ is. While this disfavors the tree semantics when we consider model-checking complexity, it advantages the tree semantics when we fix the formula. It follows from our results that in the structure semantics, fixing the formula still leaves us with the naive algorithm that checks all possible labeling for the quantified propositions. In the tree semantics, we can apply automata-theoretic methods to obtain model-checking procedures which are polynomial in the size of the Kripke structure. We can not, however, reach the space-efficient program complexity of model checking for CTL and CTL*.

We conclude with the table in Figure 4.4, summarizing our main results.

		No Quantification	Quantification with structure semantics	Quantification with tree semantics
CTL	model checking	linear time [CES86]	NP-complete [Theorem 4.5]	EXPTIME-complete [Theorem 4.6]
	program complexity	NLOGSPACE-complete [BVW94]	NP-complete [HK94, Theorem 4.11]	P-complete [Theorem 4.12]
CTL*	model checking	PSPACE-complete [EL85a]	PSPACE-complete [Theorem 4.5]	2EXPTIME-complete [Theorem 4.6]
	program complexity	NLOGSPACE-complete [BVW94]	NP-complete [Theorem 4.11]	P-complete [Theorem 4.12]

Figure 4.4: Model-checking Complexity.

4.3 Once and For All

Striving for maximality and correspondence to natural languages, philosophers developed temporal logics that provide temporal modalities that refer to both past and future [Pri57, Kam68]. Striving for minimality, computer scientists usually use temporal logics that provide only future-time modalities. Since program computations have a definite starting time and since, in this case, past-time modalities add no expressive power to linear temporal logics [GPSS80], this seems practical. Nevertheless, enriching linear temporal logics with past-time modalities makes

propositions. In this section we try to find comfort in the program complexity of the model-checking problem for these logics.

Theorem 4.11

(1) [HK94] The program complexity of EQCTL model checking is NP-complete.

(2) The program complexity of EQCTL* model checking is NP-complete.

Proof: (1) Membership in NP is immediate. In [HK94], Halpern and Kapron reduce satisfiability of CNF formulas to model checking of a fixed EQCTL formula. This establishes hardness in NP.

(2) Hardness in NP follows from the hardness for EQCTL. We prove membership in NP. In order to check whether a Kripke structure K satisfies an EQCTL* formula $\exists p_1 \ldots p_n \psi$, we guess a Kripke structure K' that differs from K in at most the labeling of $p_1 \ldots p_n$. As the program complexity of CTL* model checking is in P, the result follows. □

Thus, as long as we are interested in the structure semantics, fixing the formula brings no good news. Moreover, the fact that the program complexity of EQCTL* model checking is NP-hard implies that the PSPACE complexity we have for EQCTL* model checking is practically worse than the PSPACE complexity for CTL* model checking. Indeed, while the time complexity of the first is exponential in the Kripke structure, we have that the time complexity of the latter is exponential in the formula. Fortunately, the tree semantics (rather than the structure semantics) corresponds to the natural way branching temporal logics have been used to represent computations. There, as follows from the theorem below, the time complexity is polynomial in the Kripke structure.

Theorem 4.12 *The program complexity of both EQCTL$_t$ and EQCTL$_t^\star$ is P-complete.*

Proof: Since the algorithms given in the proof of Theorem 4.6 are polynomial in the size of K, membership in P is immediate. To prove hardness in P, we employ again the Alternating Graph Accessibility problem. We have already used its hardness in P in the proof of Theorem 3.22. Given $G, \mathcal{E}, U, s$, and t, we define $K_G = \langle \{t, exist, univ\}, V, E, s, L \rangle$, where for all $w \in \mathcal{E} \setminus \{t\}$, we have $L(w) = \{exists\}$, for all $w \in \mathcal{U} \setminus \{t\}$, we have $L(w) = \{univ\}$, and $L(t) = \{t\}$. Consider the fixed formula

$$\varphi = \exists q[q \wedge AG(q \rightarrow (t \vee (exist \wedge EXq) \vee (univ \wedge AXq))) \wedge AF\neg q].$$

The two leftmost conjunctions in φ label with q nodes of $\langle T_{K_G}, V_{K_G} \rangle$ that correspond to states $z \in V$ for which *alternating_path*(z,t) should still be verified in order to guarantee that *alternating_path*(s,t) holds. Since φ also requires that eventually no such z is left, we have that

Note that the definition of K_m^n guarantees that path quantification in $f(\psi)$ plays a role only when interpreted in states $\{1, \ldots, m\} \times \{n-1\}$. □

In fact, a more sophisticated construction can avoid the free proposition *start* (e.g., by encoding the beginning of a sequence which encodes the assignment to the atomic propositions by a sequence that does not appear elsewhere), thus showing that the EXPTIME lower bound holds even for $(1,0)$-EQCTL$_t$.

Examining our results, we conclude the following. First, in the structure semantics, existential quantification takes the model-checking problem for CTL from P to NP-complete. Thus, we can not expect an algorithm that does better than a naive check of all the possible labeling for the quantified propositions. The same penalty (moving from a deterministic complexity class to its nondeterministic variant) applies also for CTL*. There, however, as PSPACE = NPSPACE, it seems we do not really pay for it. Second, in the tree semantics, existential quantification makes model-checking as hard as satisfiability. We showed that these results hold also for very limited fragments of EQCTL and EQCTL*; e.g., when the propositional assertions are in 2CNF or when only a single quantified proposition is allowed. In addition, we showed that there are branching temporal logics $\mathcal{L}$ for which the model-checking problem for EQ$\mathcal{L}$ is harder than the satisfiability problem for $\mathcal{L}$. We now use the *finite model property* of CTL and CTL* to show that as far as satisfiability is concerned, existential quantification does not harm satisfiability complexity, for both semantics.

Theorem 4.10

(1) *The satisfiability problem for EQCTL and EQCTL$_t$ is EXPTIME-complete.*

(2) *The satisfiability problem for EQCTL* and EQCTL$_t^\star$ is 2EXPTIME-complete.*

Proof: (1) Hardness in EXPTIME follows from hardness of the satisfiability problem for CTL. To prove membership in EXPTIME, we reduce satisfiability of a formula $\varphi = \exists p_1 \ldots p_n \psi$ to the satisfiability of the CTL formula ψ. This is straightforward for φ in EQCTL, but requires some attention for φ in EQCTL$_t$. Then, while satisfaction of ψ is checked with respect to Kripke structures, satisfaction of φ is checked with respect to computation trees. It is easy to see that if ψ is satisfiable then φ is satisfiable too. For the second direction, we need the finite model property of CTL. The proof of (2) is similar, using the 2EXPTIME bounds for CTL* [VS85, ES84, EJ88]. □

4.2.3 Program Complexity of Model Checking

In the previous section, we presented some cheerless results concerning the model-checking complexity of branching temporal logics augmented with existential quantification over atomic

Note that constructing ψ above, we needed a fragment of $(1,0)$-EQCTL that contains the temporal operator EX only. The satisfiability problem for this fragment can be solved in linear time. Nevertheless, the model-checking complexity of this fragment is NP-hard. Thus, there are branching temporal logics with existential quantification for which model checking is harder than satisfiability.

Theorem 4.9 *The model-checking problem for $(1,1)$-EQCTL$_t$ is EXPTIME-hard.*

Proof: We reduce satisfiability of CTL to $(1,1)$-EQCTL$_t$ model checking. Typically, we do something similar to what we did for proving that EQCTL$_t$ model checking is EXPTIME-hard. Yet, as here we have only a single quantified proposition, we have to encode the states of K_m, as we did for the initial state in the proof of Theorem 4.8. Given $m \geq 1$ and $n \geq 1$, let $K_m^n = \langle \{start\}, W, R, w^0, L \rangle$ be the Kripke structure defined as follows:

- $W = \{1, \ldots m\} \times \{0, \ldots, n-1\}$.
- $R = \{\langle (i, n-1), (k, 0) \rangle, \langle (i,j), (i, j+1) \rangle : 1 \leq i, k \leq m, 0 \leq j \leq n-2\}$.
- $w^0 = (1,0)$.
- For all $1 \leq i \leq m$, we have $L((i,0)) = \{start\}$ and $L((i,j)) = \emptyset$ for all $j \neq 0$.

The frame of K_4^5 is presented in Figure 4.3. Now we have to translate a CTL formula $\psi(p_0, \ldots, p_{n-1})$ into a formula that instead of talking about the value of p_j at a state i of K_m, talks about the value of q at the state located j positions after the state $(i,0)$ in K_m^n. For example, the formula $EF(p_j \wedge AGp_i)$ is translated to the formula

$$EF(start \wedge (EX)^j q \wedge AG(start \rightarrow (EX)^i q)).$$

Such a translation may increase the formula ψ by at most a factor of $|\psi|$ (because of the extra EX's). Formally, we present a function f such that ψ of length m over $p_0 \ldots p_{n-1}$ is satisfiable iff $\exists q f(\psi)$ is satisfied in K_m^n. We define f by induction on the structure of ψ as follows (Q stands for either E or A):

- $f(p_i) = (EX)^i q$.
- $f(\neg \psi_1) = \neg f(\psi_1)$.
- $f(\psi_1 \vee \psi_2) = f(\psi_1) \vee f(\psi_2)$.
- $f(QX\psi_1) = (QX)^m f(\psi_1)$.
- $f(Q\psi_1 U \psi_2) = Q(strat \rightarrow f(\psi_1)) U (start \wedge f(\psi_2))$.

logic for which satisfiability is in P? Let 2CNF-EQCTL denote the subset of EQCTL in which the propositional assertions are in 2CNF.

Theorem 4.7 *The model checking problem for 2CNF-EQCTL is NP-hard.*

Proof: For every $n \geq 1$, let $\psi(n) = \bigwedge_{j \neq i} AG((\neg p_i) \vee (\neg p_j))$ where i and j range over $1 \ldots n$. For every Kripke structure K, we have that $K \models \psi(n)$ iff at most one p_i holds in each state of K. Note that all the propositional assertions in $\psi(n)$ are in 2CNF. Given a graph with n nodes, we can use $\psi(n)$ to specify properties whose decidability is NP-hard. For example, given an undirected graph $G = \langle V, E \rangle$ with $|V| = n$, let $K_G = \langle V, E', v \rangle$, where $E' = E \cup \{\langle v, v \rangle : v \in V\}$, and v is an arbitrary node in V, and let

$$\varphi = \exists p_1 \ldots p_n [\psi(n) \wedge p_1 \wedge EX(p_2 \wedge EX(p_3 \wedge \ldots \wedge EX(p_{n-1} \wedge EX(p_n \wedge EX p_1)) \cdots))].$$

It is easy to see that both K_G and φ are of size polynomial in the size of G and that $K_G \models \varphi$ iff there exists an Hamiltonian circle in G. □

Theorem 4.7 implies that it is the modality of CTL, by itself, that makes EQCTL model checking NP-hard. Still, proving the lower bounds in the theorems above, we reduced hard problems to model checking of formulas in which the number of quantified propositions is linear in the size of the reduced problem. Thus, there is still a hope that if we restrict EQCTL and EQCTL$_t$ to have a fixed number of quantified proposition, we get easier logics. The theorems below refute this hope. For $i \geq 0$ and $j \geq 0$, let (i,j)-EQCTL denote the restricted subset of EQCTL in which only i quantified propositions and j free propositions are allowed, and similarly for EQCTL$_t$.

Theorem 4.8 *The model-checking problem for* $(1,0)$*-EQCTL is NP-hard.*

Proof: We reduce SAT to $(1,0)$-EQCTL model checking. Intuitively, we do something similar to what we did for proving that EQCTL model checking is NP-hard. Since, however, a propositional formula ξ may talk about more than one proposition, we translate a formula $\xi(p_0, \ldots, p_{n-1})$ into a CTL formula that instead of talking about the value of p_i in the initial state, talks about the value of a single atomic proposition q in a state located i positions from the initial state. Formally, for $n \geq 1$, let K^n be the frame $\langle \{0, \ldots, n-1\}, R, 0 \rangle$ where $R = \{\langle 0,1 \rangle, \langle 1,2 \rangle, \ldots, \langle n-2, n-1 \rangle, \langle n-1, 0 \rangle\}$. The frame K^5 is presented in Figure 4.3. Giving a propositional formula ξ over $p_0, \ldots, p_{n-1}$, let ψ be the CTL formula obtained from replacing each occurrence of p_i in ξ by $(EX)^i q$. For example, if $\xi = (p_0 \vee p_1) \wedge (\neg p_1 \vee p_2)$, then $\psi = (q \vee EXq) \wedge (\neg EXq \vee EXEXq)$. It is easy to see that ξ is satisfiable iff $K^n \models \exists q \psi$. □

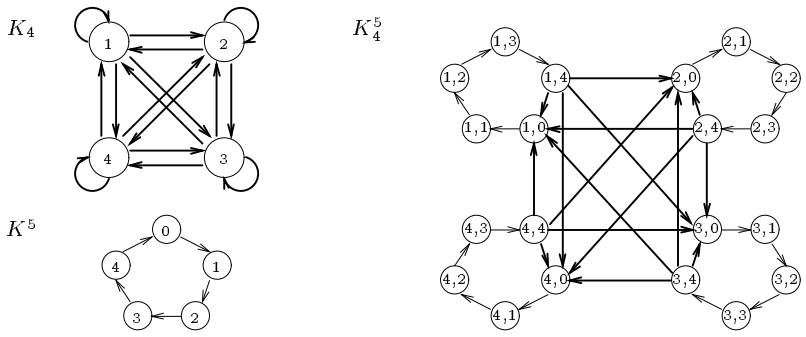


Figure 4.3: The frames K_4, K^5, and K_4^5.

Proof: (1) We first prove membership in EXPTIME. Given a set $\mathcal{D} \subset \mathbb{N}$ and an EQCTL$_t$ formula $\varphi = \exists p_1 \ldots p_n \psi$, let $\mathcal{A}_{\mathcal{D},\psi}$ be a Büchi tree automaton that accepts exactly all the $\mathcal{D}$-trees that satisfy ψ. By [VW86b], such $\mathcal{A}_{\mathcal{D},\psi}$ of size $O(|\mathcal{D}| \cdot 2^{|\psi|})$ exists. Given a Kripke structure $K = \langle AP, W, R, w^0, L\rangle$ and a set S of atomic propositions, let $\mathcal{A}_{K,S}$ be a Buchi tree automaton that accepts exactly all the $(2^{AP \cup S})$-labeled trees $\langle T_K, V'_K\rangle$ for which V'_K differs from V_K in at most the labels of the propositions in S. It is easy to see that such $\mathcal{A}_{K,S}$ of size $O(|K| \cdot 2^{|S|})$ exists. Taking $\mathcal{D}$ as the set of branching degrees in T_K and taking $S = \{p_1 \ldots p_n\}$, we get that $K \models_t \varphi$ iff $\mathcal{L}(\mathcal{A}_{K,S}) \cap \mathcal{L}(\mathcal{A}_{\mathcal{D},\psi}) \neq \emptyset$. By [VW86b], the later can be checked in time $poly(|K| \cdot 2^{|\varphi|})$.

For proving hardness in EXPTIME, we reduce the satisfiability problem for CTL, proved to be EXPTIME-hard in [FL79], to EQCTL$_t$ model checking. For $m \geq 1$, let K_m be the frame $\langle \{1, \ldots, m\}, \{1, \ldots, m\} \times \{1, \ldots, m\}, 1\rangle$. The frame K_4 is presented in Figure 4.3. Since a CTL formula ψ is satisfiable iff it is satisfied in a tree of branching degree $|\psi|$, and since unwinding $K_{|\psi|}$ results in such a tree, satisfiability of ψ can be reduced to model checking of $K_{|\psi|}$ with respect to the EQCTL$_t$ formula $\exists p_1 \ldots p_n \psi$, where $p_1 \ldots p_n$ are exactly all the atomic propositions in ψ.

(2) The model-checking procedure for EQCTL$_t^\star$ is similar to the one for EQCTL$_t$. Here, following [ES84], we have that $\mathcal{A}_{\mathcal{D},\psi}$ is a Rabin tree automaton with $2^{2^{|\psi|}}$ states and $2^{|\psi|}$ pairs. By [EJ88], checking the nonemptiness of $\mathcal{L}(\mathcal{A}_{K,S}) \cap \mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ can then be done in time $poly(|K| \cdot 2^{2^{|\varphi|}})$. To prove hardness of EQCTL$_t^\star$ model checking in 2EXPTIME, we reduce satisfiability of CTL*, proved to be 2EXPTIME-hard in [VS85], to EQCTL$_t^\star$ model checking. Since a CTL* formula ψ is satisfiable iff it is satisfied in a tree of branching degree $|\psi|$, the same reduction that works for EQCTL$_t$ works also here. □

As CTL subsumes propositional logic, being EQCTL model checking NP-hard is far from being surprising. What, however, if we restrict CTL to subsume only a subset of propositional

existential quantification is sufficient to express any occurrence of events in the past that can be expressed by linear temporal logic. In addition, we can use existential quantification to count y modulo z. The way we use formulas in the structure semantics is different. There, formulas describe a single computation which is a partially ordered set [PW84]. For example, the formula $\exists q(q \wedge AG(q \rightarrow AXAXq) \wedge AG(q \rightarrow send_i)$ specifies that process i sends a massage in all its even positions. So, existential quantification is definitely helpful. We now turn to analyze the complexity of the model-checking problem and consider its price.

4.2.2 Model-Checking Complexity

The model-checking problem for a variety of branching temporal logics can be stated as follows: given a branching temporal logic formula φ and a finite Kripke structure $K = \langle AP, W, R, w^0, L\rangle$, determine whether K satisfies φ. When some of the logics are sensitive to unwinding, there are two possible interpretations of this problem. The first interpretation, which is the one appropriate for EQCTL and EQCTL*, asks whether $K \models \varphi$. In the second interpretation, which is the one appropriate for EQCTL$_t$ and EQCTL$_t^\star$, we are given φ and K and are asked to determine whether $K \models_t \varphi$. In this section we consider model-checking complexity for the two interpretations.

Theorem 4.5

(1) *The model-checking problem for EQCTL is NP-complete.*

(2) *The model-checking problem for EQCTL* is PSPACE-complete.*

Proof: (1) We first prove membership in NP. In order to check whether a Kripke structure K satisfies an EQCTL formula $\exists p_1 \ldots p_n \psi$, we guess a Kripke structure K' that differs from K in at most the labeling of $p_1 \ldots p_n$, and then check, in linear time [CES86], whether K' satisfies the CTL formula ψ. To prove hardness in NP, we do a reduction from SAT. Clearly, a propositional formula ξ over the propositions $p_1 \ldots p_n$ is satisfiable if and only if the EQCTL formula $\exists p_1 \ldots p_n \xi$ is satisfied in a one-state frame.

(2) Both membership and hardness in PSPACE follow from being CTL* model checking PSPACE-complete [EL85a]. While hardness is immediate, Savitch Theorem [Sav70] is required for the membership. □

Theorem 4.6

(1) *The model-checking problem for EQCTL$_t$ is EXPTIME-complete.*

(2) *The model-checking problem for EQCTL$_t^\star$ is 2EXPTIME-complete.*

$p \in AP \setminus \{p_1, \ldots, p_n\}$, we have $p \in V_K(x)$ iff $p \in V'_K(x)$. Note that $K \models \varphi$ implies that $K \models_t \varphi$. It is the other direction which makes EQCTL sensitive to unwinding.

An interesting example for the sensitivity of EQCTL to unwinding is the formula $\varphi_1 = \exists q(q \wedge (AX\neg q) \wedge AG(q \leftrightarrow AXAXq) \wedge AG(q \rightarrow p))$. The formula is suggested in the literature for specifying the property $G2(p)$ ="p holds in all even places". When interpreted over computation trees, φ_1 indeed specifies $G2(p)$. Yet, for a Kripke structure with a state that can be reached from the initial state by both an even number and an odd number of transitions (e.g., a Kripke structure that consists of a single state with a self loop), any labeling of q fails, even if this Kripke structure does satisfy $G2(p)$. Hence, φ_1 is appropriate only for the tree semantics.

Insensitivity to the sensitivity of EQCTL and EQCTL* to unwinding hides also when comparing these logics with tree automata [ES84]. Indeed, EQCTL$^\star_t$ is as expressive as symmetric pair automata on infinite binary trees. Nevertheless, the translation of EQCTL$^\star_t$ into $2S2$, which is the base of this equivalence, does not hold for EQCTL*. Similarly, it is EQCTL$_t$, only, which is as expressive as symmetric Büchi automata on infinite binary trees.

We have just seen that EQCTL$_t$ is strong enough to specify $G2(p)$. In fact, the formula $\varphi_2 = \exists q(q \wedge AG(q \rightarrow AXAXq) \wedge AG(q \rightarrow p))$ specifies $G2(p)$ faithfully with respect to both the tree and the structure semantics. As opposed to φ_1, the formula φ_2 enables states which can be reached from the initial state by both an even and an odd number of transitions can be labeled with q. As CTL can not specify $G2(p)$ [Wol83], we have the following:

Theorem 4.4 *EQCTL and EQCTL$_t$ are both strictly more expressive than CTL.*

Theorems 4.3 and 4.4 clearly hold also with respect to EQCTL*. The increase in the expressive power is significant. In particular, existential quantification in the tree semantics is strong enough to replace *satellites*. A satellite, as introduced in [BBG+94], is a small finite state machine, linked to a design to be verified. It can read the design at any moment and it records particular events of interest, for possible use in the specification of the design. A concept similar to satellites is introduced in [Lon93] as *observer processes*. For example, we can define a satellite *Raise*(s) which detects cycles in which the signal s is raised (changes its value from ***false*** to ***true***). Satellites overcome the expressiveness limitations of CTL and are used successfully as a part of the formal-verification system in IBM Haifa. The price of satellites is the increase in the state space, which now consists of the product of the state space of the design with the state space of the satellite. Existential quantification leaves the design clean and shifts this price to the specification. For example, instead of checking a CTL formula ψ which requires the activation of the satellite *Raise*(s), we can check the EQCTL$_t$ formula obtained from ψ by conjuncting it with $\exists qAG(s \rightarrow AXq) \wedge AG(\neg s \rightarrow AX\neg q)$ and replacing each occurrence of *Raise*(s) by $s \wedge \neg q$. Note that the quantified proposition q labels a node iff s holds in its predecessor node. In fact, in Section 4.3 we are going to see that

is defined similarly, by adding existential quantification to CTL.

Given a formula $\exists p_1 \ldots p_n \psi$, we call the atomic propositions $p_1 \ldots p_n$ *quantified propositions* and we call all the other propositions in ψ *free propositions*. Note that satisfaction of an EQCTL* formula with no free propositions in a Kripke structure K is independent of AP and L. A *frame* is a Kripke structure with no AP and L. A frame $K = \langle W, R, w^0 \rangle$ satisfies an EQCTL* formula $\exists p_1 \ldots p_n \psi$ iff there exists a Kripke structure $K' = \langle AP, W, R, w^0, L \rangle$ with $\{p_1, \ldots, p_n\} \subseteq AP$, such that $K' \models \psi$.

Consider a Kripke structure K and the computation tree $\langle T_K, V_K \rangle$ obtained by unwinding K. Each state in K may correspond to several nodes in $\langle T_K, V_K \rangle$. Since all these nodes have the same future (i.e., they root identical subtrees) and since CTL can refer only to the future, CTL is *insensitive to unwinding*. That is, for every CTL formula φ and for every Kripke structure K, we have that $K \models \varphi$ iff $\langle T_K, V_K \rangle \models \varphi$. Insensitivity to unwinding is an important property for a branching temporal logic. Logics which are insensitive to unwinding, we can model check their formulas with respect to a finite Kripke structure, and adopt the result for its infinite computation tree. Symmetrically, we can model check an infinite computation tree using, say, automata-theoretic methods, and adopt the result for all Kripke structures that can be unwound into this tree. In Theorem 4.3 below we show that once we augment CTL with existential quantification over atomic propositions, it becomes sensitive to unwinding.

Theorem 4.3 *EQCTL is sensitive to unwinding.*

Proof: Consider the EQCTL formula $\varphi = \exists q AG(p \leftrightarrow AXq)$ and consider the Kripke structure

$$K = \langle \{p\}, \{w_0, w_1\}, \{\langle w_0, w_1 \rangle, \langle w_1, w_1 \rangle\}, w_0, \{\langle w_0, \{p\} \rangle, \langle w_1, \emptyset \rangle\} \rangle.$$

Since $p \in L(w_0)$ and since w_1 is a successor of w_0, it must be that q holds in the state w_1 of a Kripke structure that satisfies $AG(p \leftrightarrow AXq)$ and is $\{q\}$-different from K. On the other hand, since $p \notin L(w_1)$ and and since w_1 is the only successor of itself, it must be that q does not hold in the state w_1 of a Kripke structure that satisfies $AG(p \leftrightarrow AXq)$ and is $\{q\}$-different from K. Thus, there exists no Kripke structure that satisfies $AG(p \leftrightarrow AXq)$ and is $\{q\}$-different from K. Hence, $K \not\models \varphi$. We now show that $\langle T_K, V_K \rangle \models \varphi$. Consider the computation tree $\langle T_K, V'_K \rangle$ over the alphabet $2^{\{p,q\}}$, where $V'_K(0) = \{p\}$, $V'_K(1) = \{q\}$, and for all $x \geq 2$, we have that $V'_K(x) = \emptyset$. Clearly, $\langle T_K, V'_K \rangle \models AG(p \leftrightarrow AXq)$ and thus, $\langle T_K, V_K \rangle \models \varphi$. □

So, it makes sense to define two different semantics for EQCTL. The first corresponds to the original structure semantics and the second, which we call EQCTL$_t$, corresponds to a tree semantics. Precisely, an EQCTL$_t$ formula $\varphi = \exists p_1 \ldots p_n \psi$ is satisfied in a Kripke structure K, denoted $K \models_t \varphi$, iff there exists a computation tree $\langle T_K, V'_K \rangle$ such that $\langle T_K, V'_K \rangle \models \psi$ and V'_K differs from V_K in at most the labeling of $p_1, \ldots, p_n$; i.e., for every $x \in T_K$ and for every

the other hand, as in satisfiability, we are asked about the existence of some Kripke structure that satisfies the formula. Essentially, we can view the model-checking problem for EQ$\mathcal{L}$ as a restricted (or perhaps extended) version of the satisfiability problem for $\mathcal{L}$, in which the candidates to satisfy the formula are not all Kripke structures, but only a limited subset of them. Here, naturally, comes the question of complexity. The satisfiability problem for a branching temporal logic $\mathcal{L}$ is usually harder than its model-checking problem. For example, we have seen that the branching temporal logics CTL and CTL* have, respectively, EXPTIME and 2EXPTIME complete satisfiability bounds [FL79, VS85, ES84, EJ88] and have, respectively, linear-time and PSPACE-complete bounds for their model checking problems [CES86, EL85a]. Where does the complexity of the model-checking problem for EQ$\mathcal{L}$ stand? Is it necessarily between the complexities of the model-checking problem and the satisfiability problem for $\mathcal{L}$? To which of them is it closer? Is it worth paying the increase in model-checking complexity for the increase in the expressive power?

A key observation that should be made before answering these questions is that once we add existential quantification to a branching temporal logic $\mathcal{L}$, it becomes *sensitive to unwinding*. That is, unwinding of a Kripke structure into an infinite computation tree does not preserve the set of EQ$\mathcal{L}$ formulas it satisfies. Consequently, we distinguish between two semantics for EQ$\mathcal{L}$. The first is the structure semantics given above. The second, which we call EQ$\mathcal{L}_t$, corresponds to a tree semantics. According to this semantics, a Kripke structure K satisfies a formula $\exists p_1 \ldots p_n \psi$ iff there exists a computation tree that satisfies ψ and differs from the computation tree obtained by unwinding K in at most the labels of $p_1, \ldots, p_n$. Intuitively, it is harder for K to satisfy a formula in the structure semantics: among the infinitely many computation trees that we have as candidates for satisfaction in the tree semantics, only finitely many (these in which nodes that correspond to the same state of K have the same labeling) are candidates in the structure semantics. We show that the logics EQ$\mathcal{L}$ and EQ$\mathcal{L}_t$ differ in their practices as specification languages, differ in their expressive power, and differ in their model-checking complexities. Nevertheless, we found in the literature unawareness to this sensitivity.

4.2.1 Expressiveness

The logic *EQCTL** is obtained by adding existential quantification to CTL*. If ψ is a CTL* formula and $p_1 \ldots p_n$ are atomic propositions, then $\exists p_1 \ldots p_n \psi$ is an EQCTL* formula. Given two Kripke structures $K = \langle AP, W, R, w^0, L\rangle$ and $K' = \langle AP', W', R, w'^0, L'\rangle$, we say that K' is $\{p_1, \ldots, p_n\}$-*different* from K iff $AP' = AP \cup \{p_1, \ldots, p_n\}$, $W' = W$, $R' = R$, $w'^0 = w^0$, and for all $w \in W$ and $p \in AP \setminus \{p_1, \ldots, p_n\}$, we have that $p \in L'(w)$ iff $p \in L(w)$. The semantics of $\exists p_1 \ldots p_n \psi$ is given by $K \models \exists p_1 \ldots p_n \psi$ iff there exists a Kripke structure K', such that $K' \models \psi$ and K' is $\{p_1, \ldots, p_n\}$-different from K. Note that EQCTL* is not closed under negation. Thus, formulas of the form $\forall p_1 \ldots p_n \psi$ are not EQCTL* formulas. The logic *EQCTL*

4.1.4 The Logics CTL^i

For every $i \geq 1$, let us define the subset CTL^i of CTL*. A CTL* formula ψ is in CTL^i if and only if each of its path formulas contains at most i operators. The logic CTL, for instance, is syntactically equivalent to CTL$_1$. Consider the construction of HAAs presented for CTL* formulas in Theorem 3.14. The exponential blow up in these HAAs originates from formulas of the form $E\xi$. Hence, bounding the number of operators in the path formulas bounds the exponential factor in the construction. Indeed, composing automata for φ_1 and φ_2 into an automaton for either $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ causes no blow up. Precisely, an HAA associated with a CTL^i formula ψ is of size $O(|\psi| \cdot 2^i)$. Since the 1-letter nonemptiness problem for HAAs is in linear time, this implies that model checking of a CTL^i formula ψ in a Kripke structure K can be done in time $O(|K| \cdot |\psi| \cdot 2^i)$. The above discussion can be viewed an the automata-theoretic counterpart to the reduction of branching-time model checking to linear-time model checking in [EL85a].

Thus, in particular, the fact that we were able to provide a linear-time model-checking procedure for CTL^2 is not surprising. Today's model-checking tools involve many heuristics and methods which make the theoretical complexity bounds only one factor, among many others, when considering the adaption of a new model-checking tool. Hence, when we improve model-checking complexity, a crucial step is to check how the new procedures combine with existing methods. The advantage of our procedure is that once having a CTL model-checking package, extending it to a one that handles CTL^2 is very easy. Indeed, for the happy subset of CTL^2, it involves only a translation. For the rest of CTL^2, techniques like BDDs [BCM+92] that are helpful in CTL model checking, are adaptable for CTL^2. In addition, the extension preserves the ability to handle fairness. Hence, we believe that CTL^2 constitutes an additional step in the process of making model checking a practical and convenient tool for formal verification.

4.2 Existential Quantification over Atomic Propositions

Adding *quantification over atomic propositions* increases the expressive power of temporal logics [Sis83, Wol83, SVW87, PR89]. In this section, based on [Kup95], we consider the extension of branching temporal logics with *existential* quantification. Formally, if ψ is a formula in some branching temporal logic $\mathcal{L}$, then $\exists p_1 \ldots p_n \psi$, where $p_1, \ldots, p_n$ are atomic propositions, is a formula in the logic EQ$\mathcal{L}$, which augments $\mathcal{L}$ with existential quantification. The formula $\exists p_1 \ldots p_n \psi$ is satisfied in a Kripke structure K iff there exists a Kripke structure that satisfies ψ and differs from K in at most the labeling of $p_1, \ldots, p_n$.

The model-checking problem for EQ$\mathcal{L}$ stands somewhere between the model-checking and the satisfiability problems for $\mathcal{L}$. On the one hand, as in model checking, we are given both a Kripke structure and a formula and we are asked whether the structure satisfies the formula. On

```
procedure EGU_check (φ2, φ3, φ);
begin
  for j ∈ [0 ... |W|] do S^j := ∅ end;
  for every C ∈ find_MSCCs(φ2 ∨ φ3) do
      if there exists w ∈ C such that φ3 ∈ L(w) then
        for every w ∈ C do add φ to L(w); add w to S^0 end
  end;
  j := 0;
  while j < |W| do
    S := S^j;
    while S ≠ ∅ do
      remove some w from S;
      for every predecessor w' of w do
          if φ ∉ L(w') and (φ2 ∈ L(w') or φ3 ∈ L(w')) then
            add φ to L(w') ; add w' to S^{j+1}
      end;
    end;
    j := j + 1;
  end;
end;
```

Figure 4.2: The procedure *EGU_check*.

A crucial point in our extension is that the equivalences given for happy formulas are valid also when interpreting the formulas over modules. Therefore, the only thing to do is to extend our *EGU_check* procedure to handle fairness. This can easily be done by requiring each MSCC in $\mathit{find\_MSCCs}(\varphi_2 \vee \varphi_3)$ to contain at least one state from each $F \in \alpha$. In fact, having a fair model-checking procedure for CTL, things are even simpler. There, one can check the formula $EG(\varphi_2 U \varphi_3)$ by checking the CTL formula $EG(\varphi_2 \vee \varphi_3)$ with a fairness condition φ_3 [Gei94].

As in CTL, fairness constraints can be used to increase the expressive power of the logic. Instead of a set of subsets of W, one can specify the fairness condition α as a set of CTL^2 formulas. A path π is fair iff for each formula $\xi \in \alpha$, there are infinitely many states in π that satisfy ξ. Consider the CTL* formula $\psi = A(GFp \rightarrow \varphi)$, where p is atomic. Checking the formula $A\varphi$ with respect to a module that has $\alpha = \{\{p\}\}$ as its fairness condition is equivalent to checking ψ in a Kripke structure with no fairness conditions. Using the extended model checker for CTL^2, we can thus check formulas of the form $A(GFp \rightarrow \varphi)$ for which $A\varphi$ is a CTL^2 formula. In particular, we can check the formula $A(GFp \rightarrow GFq)$ that specifies strong fairness. So, the combination of CTL^2 and fairness is really powerful.

```
procedure model_check (ψ, K);
begin
  for i := 1 to |cl(ψ)| do
      for every φ ∈ cl(ψ) with |cl(φ)| = i do
          case structure of φ is of the form
              An atomic proposition : noop;
              φ1 ∨ φ2 : for every w ∈ W do
                            if φ1 ∈ L(w) or φ2 ∈ L(w) then add φ1 ∨ φ2 to L(w);
              ¬φ1     : for every w ∈ W do
                            if φ1 ∉ L(w) then add ¬φ1 to L(w);
              EXφ1    : for every w ∈ W do
                            if φ1 ∈ L(w') for some successor w' of w then add EXφ1 to L(w);
              Eφ1Uφ2 : EU_check (φ1, φ2, φ);
              Eφ1Ũφ2 : EŨ_check (φ1, φ2, φ);
              Eφ1Ũ(φ2Uφ3) : for every w ∈ W do
                            if E(φ2 ∨ φ3)U(φ1 ∧ E(φ2Uφ3)) ∈ L(w) and EG(φ2Uφ3) ∈ L(w) then
                              add Eφ1Ũ(φ2Uφ3) to L(w);
              EG(φ2Uφ3) : EGU_check (φ2, φ3, φ)
          end;
end;
```

Figure 4.1: The model-checking procedure for CTL^2.

to non-trivial MSCC. I.e., a single state without a self loop is not a MSCC). The output of *find_MSCCs*(ψ) is a set of (disjoint) subsets of W such that every subset is a MSCC.

We now consider the complexity of the procedure. As *find_MSCCs*(ψ) is of complexity linear in $|K|$ [Tar72], initialization is performed in linear time. Since all the sets S^j are pairwise disjoint, the body of the for-loop is executed at most once for each transition in R. Hence, the whole procedure terminates after at most $O(|K|)$ steps.

Fair Model Checking for CTL^2

In Section 3.6, we considered modules and the fair model-checking problem. There, each module has a Rabin acceptance condition α. In this section, we consider modules in which $\alpha \subseteq 2^W$ is a set of subsets of W, each corresponds to a Büchi acceptance condition. A path $\pi = w_0, w_1, \ldots$ is fair iff for each set $F \in \alpha$ we have $w_i \in F$ for infinitely many $i \geq 0$. Such a fairness condition was considered in [CES86]. Clarke et al. show that the fair model checking problem for CTL can be solved in linear time. Todays model-checking tools for CTL handle modules with this fairness condition. We now show that once one has a fair model-checking tool for CTL, it can be very easily extended to handle CTL^2, preserving its complexity.

Below we define the *closure of a formula* for every CTL^2 formula. The closure of φ, $cl(\varphi)$, extends the notion of sub-formulas used in [CES86]. Intuitively, it is the set of CTL^2 formulas that contains φ and all the formulas that the labeling of φ depends on. Given a CTL^2 formula φ, the set $cl(\varphi)$ is inductively defined as follows (φ_1, φ_2, and φ_3 are CTL^2 formulas):

- If φ is either ***true*** or ***false***, then $cl(\varphi) = \{\varphi\}$.
- If φ is a proposition, then $cl(\varphi) = \{\varphi, \textbf{\textit{true}}, \textbf{\textit{false}}\}$.
- If φ is $\neg\varphi_1$ or $EX\varphi_1$, then $cl(\varphi) = \{\varphi\} \cup cl(\varphi_1)$.
- If φ is $\varphi_1 \vee \varphi_2$, $E\varphi_1 U \varphi_2$, or $E\varphi_1 \tilde{U} \varphi_2$, then $cl(\varphi) = \{\varphi\} \cup cl(\varphi_1) \cup cl(\varphi_2)$.
- If φ is a happy formula, then $cl(\varphi) = cl(\varphi')$, where φ' is the CTL equivalent of φ.
- If φ is $E\varphi_1 \tilde{U} (\varphi_2 U \varphi_3)$, then $cl(\varphi) = \{\varphi, EG(\varphi_2 U \varphi_3)\} \cup cl(E(\varphi_2 \vee \varphi_3) U (\varphi_1 \wedge E(\varphi_2 U \varphi_3)))$.

Lemma 4.2 *For every CTL^2 formula ψ,*

(a) *All the formulas in $cl(\psi)$ are either* ***true***, ***false***, *atomic propositions, or have the form $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $EX\varphi_1$, $E\varphi_1 U \varphi_2$, $E\varphi_1 \tilde{U} \varphi_2$, $E\varphi_1 \tilde{U} (\varphi_2 U \varphi_3)$, or $EG(\varphi_2 U \varphi_3)$.*

(b) *The size of $cl(\psi)$ is linear in $|\psi|$.*

Lemma 4.2 is the key for the model-checking procedure, presented in Figure 4.1.

Given a CTL^2 formula ψ, the procedure is called with the CTL equivalent of ψ (if exists). As in [CES86], it iteratively labels formulas from $cl(\varphi)$. It is guaranteed that when φ is labeled, then all the formulas in $cl(\varphi) \setminus \{\varphi\}$ are already labeled. Formulas of the form $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $EX\varphi_1$, $E\varphi_1 U \varphi_2$, or $E\varphi_1 \tilde{U} \varphi_2$ are labeled using the corresponding procedure described in [CES86]. Formulas of the form $E\varphi_1 \tilde{U} (\varphi_2 U \varphi_3)$, are labeled, according to Lemma 4.1, using the procedure *EGU_Check*$(\varphi_2, \varphi_3, \varphi)$, presented in Figure 4.2 and explained below.

EGU_check$(\varphi_2, \varphi_3, \varphi)$ labels states w of K for which $w \models EG\varphi_2 U \varphi_3$. Note that $w \models EG\varphi_2 U \varphi_3$ iff there exists a path π starting at w such that all the states in π satisfy either φ_2 or φ_3, and infinitely many states in π satisfy φ_3. Accordingly, *EGU_check* first labels states that belong to a maximal strongly connected component (MSCC) for which all states are labeled by either φ_2 or φ_3, and there exists a state in it that satisfies φ_3. These states are inserted into the set S^0, meaning that there is a path of length 0 from them to such a MSCC. Then, the procedure proceeds backwards the transition relation, inserting to S^j states for which there exists a path (whose all states satisfy either φ_2 or φ_3) of length j from them to such a MSCC. Clearly, there is no need to proceed with j's greater than $|W|$.

EGU_check uses the procedure *find_MSCCs*(ψ), which, given a formula ψ, finds the MSCCs in the sub-structure of K that consists of all the states in K that are labeled with ψ (we refer

Hence, it is left to show that $E((\varphi_2 U \varphi_3) U (\varphi_1 \wedge (\varphi_2 U \varphi_3))) \sim E((\varphi_2 \vee \varphi_3) U (\varphi_1 \wedge E(\varphi_2 U \varphi_3)))$.

It is easy to see that a state that satisfies the left-hand side of the equivalence, satisfies also its right-hand side. We consider the second direction. Assume that $w \models E(\varphi_2 \vee \varphi_3) U (\varphi_1 \wedge E\varphi_2 U \varphi_3)$. Then, there exists a path $\pi = w_0, w_1, \ldots$, with $w_0 = w$, for which there exists $i \geq 0$ such that $w_i \models \varphi_1 \wedge E\varphi_2 U \varphi_3$, and for all $0 \leq j < i$, we have that $w_j \models \varphi_2 \vee \varphi_3$. If $w_i \models \varphi_3$, then it is guaranteed that every suffix π^j of π eventually reaches a state (w_i or an earlier one) that satisfies φ_3. Also, as for all $0 \leq j < i$, we have that $w_j \models \varphi_2 \vee \varphi_3$, then, as long as φ_3 is not reached, φ_2 is satisfied. Therefore, in this case, $\pi \models (\varphi_2 U \varphi_3) U (\varphi_1 \wedge (\varphi_2 U \varphi_3))$. If $w_i \not\models \varphi_3$, then there must exist a path $\pi' = w_i', w_{i+1}', \ldots$, with $w_i' = w_i$ such that $\pi' \models \varphi_2 U \varphi_3$. Consider the path $\rho = w_0, w_1, \ldots, w_i, w_{i+1}', w_{i+2}', \ldots$. Similarly to the previous case, $\rho \models (\varphi_2 U \varphi_3) U (\varphi_1 \wedge (\varphi_2 U \varphi_3))$ and thus $w \models E(\varphi_2 U \varphi_3) U (\varphi_1 \wedge (\varphi_2 U \varphi_3))$. □

In the next section we show that the additional expressive power of CTL^2 is given for free. Namely, the model-checking problem for the logic CTL^2 is of the same complexity as is the one for CTL.

4.1.3 Model Checking

In this section we introduce an efficient algorithm for the model-checking problem for CTL^2. Given a Kripke structure $K = \langle AP, W, R, w^0, L \rangle$ and a CTL^2 formula ψ, our algorithm labels K such that for every state $w \in W$, we have that w is labeled with ψ iff $w \models \psi$. The algorithm is of complexity linear in both the size of K and the length of ψ. Thus, the increase in the expressive power does not effect model-checking complexity which remains linear, exactly as the one for CTL.

The algorithm extends the efficient model checker for CTL introduced in [CES86]. There, a formula is handled by successively applying a state labeling algorithm to its sub-formulas. In more details, given a CTL formula ψ and a Kripke structure K, the algorithm takes the sub-formulas of ψ, starting with the innermost ones, and, iteratively, labels with each sub-formula exactly these states of K that satisfy it. Each iteration handles a single sub-formula which may have one of seven forms that together cover CTL modalities. Since ψ has at most $|\psi|$ sub-formulas, the check terminates after at most $|\psi|$ iterations. Each iteration requires $O(|K|)$ steps and hence, the entire check is accomplished after $O(|\psi| * |K|)$ steps.

In CTL^2, there are more than seven forms. All the possible combinations of one or two temporal operators should be considered. However, as we showed in the previous section, all those forms, except $E\varphi_1 \tilde{U} (\varphi_2 U \varphi_3)$, have CTL equivalences. Thus, labeling with happy sub-formulas can be done by labeling their CTL equivalences. In addition, we introduce a procedure that handles sad sub-formulas.

3. $E((\varphi_1 U \varphi_2) \vee (\varphi_3 U \varphi_4)) \sim (E\varphi_1 U \varphi_2) \vee (E\varphi_3 U \varphi_4)$
$E((\varphi_1 U \varphi_2) \vee (\varphi_3 \tilde{U} \varphi_4)) \sim (E\varphi_1 U \varphi_2) \vee (E\varphi_3 \tilde{U} \varphi_4)$
$E((\varphi_1 \tilde{U} \varphi_2) \vee (\varphi_3 \tilde{U} \varphi_4)) \sim (E\varphi_1 \tilde{U} \varphi_2) \vee (E\varphi_3 \tilde{U} \varphi_4)$

4. $E((\varphi_1 U \varphi_2) \wedge (\varphi_3 U \varphi_4)) \sim E((\varphi_1 \wedge \varphi_3) U (\varphi_2 \wedge E\varphi_3 U \varphi_4)) \vee E((\varphi_1 \wedge \varphi_3) U (\varphi_4 \wedge E\varphi_1 U \varphi_2))$
$E((\varphi_1 U \varphi_2) \wedge (\varphi_3 \tilde{U} \varphi_4)) \sim E((\varphi_2 \wedge \varphi_3) U (\varphi_4 \wedge E\varphi_1 \tilde{U} \varphi_2)) \vee E((\varphi_2 \wedge \varphi_3) U (\varphi_2 \wedge E\varphi_3 \tilde{U} \varphi_4))$
$E((\varphi_1 \tilde{U} \varphi_2) \wedge (\varphi_3 \tilde{U} \varphi_4)) \sim E((\varphi_2 \wedge \varphi_4) U (\varphi_3 \wedge E\varphi_1 \tilde{U} \varphi_2)) \vee E((\varphi_2 \wedge \varphi_4) U (\varphi_1 \wedge E\varphi_3 \tilde{U} \varphi_4)) \vee EG(\varphi_2 \wedge \varphi_4)$

Equivalences for nested temporal operators

1. $EXX\varphi_1 \sim EXEX\varphi_1$
$EX\varphi_1 U \varphi_2 \sim EXE\varphi_1 U \varphi_2$
$EX\varphi_1 \tilde{U} \varphi_2 \sim EXE\varphi_1 \tilde{U} \varphi_2$

2. $E\varphi_1 \tilde{U}(X\varphi_2) \sim EX\varphi_2 \wedge (\varphi_1 \vee EXE\varphi_1 \tilde{U}(\varphi_2 \wedge EX\varphi_2))$
$E\varphi_1 \tilde{U}(\varphi_2 \tilde{U} \varphi_3) \sim E\varphi_1 \tilde{U}(E\varphi_2 \tilde{U} \varphi_3)$

3. $E(X\varphi_1)\tilde{U}\varphi_2 \sim E(EX\varphi_1)\tilde{U}\varphi_2$
$E(\varphi_1 \tilde{U} \varphi_2)\tilde{U}\varphi_3 \sim E(E\varphi_1 \tilde{U} \varphi_2)\tilde{U}\varphi_3$
$E(\varphi_1 U \varphi_2)\tilde{U}\varphi_3 \sim E(E\varphi_1 U \varphi_2)\tilde{U}\varphi_3$

4. $E\varphi_1 U(X\varphi_2) \sim E\varphi_1 U(EX\varphi_2)$
$E\varphi_1 U(\varphi_2 U \varphi_3) \sim E\varphi_1 U(E\varphi_2 U \varphi_3)$
$E\varphi_1 U(\varphi_2 \tilde{U} \varphi_3) \sim E\varphi_1 U(E\varphi_2 \tilde{U} \varphi_3)$

5. $E(X\varphi_1)U\varphi_2 \sim \varphi_2 \vee EXE\varphi_1 U(\varphi_2 \wedge \varphi_1)$
$E(\varphi_1 U \varphi_2)U\varphi_3 \sim \varphi_3 \vee E(\varphi_1 \vee \varphi_2)U((\varphi_2 \wedge EX\varphi_3) \vee (\varphi_3 \wedge E\varphi_1 U \varphi_2))$
$E(\varphi_1 \tilde{U} \varphi_2)U\varphi_3 \sim \varphi_3 \vee E\varphi_2 U((\varphi_1 \wedge \varphi_2 \wedge EX\varphi_3) \vee (\varphi_3 \wedge E\varphi_1 \tilde{U} \varphi_2))$

We now show that sad formulas have equivalences in CTL augmented with $EG(\varphi_2 U \varphi_3)$.

Lemma 4.1 $E\varphi_1 \tilde{U}(\varphi_2 U \varphi_3) \sim E((\varphi_2 \vee \varphi_3)U(\varphi_1 \wedge E(\varphi_2 U \varphi_3))) \vee EG(\varphi_2 U \varphi_3)$.

Proof: Consider the congruence

$$\varphi_1 \tilde{U} \psi \approx (\psi U(\varphi_1 \wedge \psi)) \vee G\psi.$$

Taking $\psi = \varphi_2 U \varphi_3$, we get $E\varphi_1 \tilde{U}(\varphi_2 U \varphi_3) \sim E(((\varphi_2 U \varphi_3)U(\varphi_1 \wedge (\varphi_2 U \varphi_3))) \vee G(\varphi_2 U \varphi_3))$. By distributing the E quantifier over the disjunction, we have that

$$E\varphi_1 \tilde{U}(\varphi_2 U \varphi_3) \sim E((\varphi_2 U \varphi_3)U(\varphi_1 \wedge (\varphi_2 U \varphi_3))) \vee EG(\varphi_2 U \varphi_3)).$$

$\varphi_1 = invoke(p)$ and $\varphi_2 = \neg execute(p)$, gives "in all computation paths, if p is executed, then p is invoked two steps earlier or before". Note that since we usually want to specify behaviors as invariants, we should precede these formulas with AG, resulting in formulas like $AGA\varphi_1 U(\varphi_2 \tilde{U} \varphi_3)$ or $AGE(\varphi_1 \tilde{U}(X\varphi_2))$, all syntactically correct in CTL^2

Surprisingly, the increase in the expressive power of CTL^2 with respect to CTL is not as significant as one might expect. In the rest of this section we show that the CTL^2 formula $EG(\varphi_2 U \varphi_3)$ embodies all the expressiveness superiority of CTL^2 over CTL. Namely, extending CTL by the binary operator EGU results in a logic which is as expressive as CTL^2! To do this, we first present the logic CTL^2 in terms of modalities. It is easy to see that each CTL^2 formula has one of the following forms (where $*$ stands for either $\vee$ or $\wedge$ and φ_1, φ_2, φ_3, and φ_4 are CTL^2 formulas):

R1. ***true***, ***false*** or p, for all $p \in AP$.

R2. $\neg\varphi_1$ or $\varphi_1 \vee \varphi_2$.

R3. $EX\varphi_1$, $E\varphi_1 U \varphi_2$, or $E\varphi_1 \tilde{U} \varphi_2$.

R4. $E(\varphi_1 * (X\varphi_2))$, $E(\varphi_1 * (\varphi_2 U \varphi_3))$, $E(\varphi_1 * (\varphi_2 \tilde{U} \varphi_3))$,
$E((X\varphi_1) * (X\varphi_2))$, $E((X\varphi_1) * (\varphi_2 U \varphi_3))$, $E((X\varphi_1) * (\varphi_2 \tilde{U} \varphi_3))$,
$E((\varphi_1 U \varphi_2) * (\varphi_3 U \varphi_4))$, $E((\varphi_1 U \varphi_2) * (\varphi_3 \tilde{U} \varphi_4))$, or $E((\varphi_1 \tilde{U} \varphi_2) * (\varphi_3 \tilde{U} \varphi_4))$.

R5. $EXX\varphi_1$, $EX(\varphi_1 U \varphi_2)$, $EX(\varphi_1 \tilde{U} \varphi_2)$,
$E((X\varphi_1) U \varphi_2)$, $E((\varphi_1 U \varphi_2) U \varphi_3)$, $E((\varphi_1 \tilde{U} \varphi_2) U \varphi_3)$,
$E(\varphi_1 U (X\varphi_2))$, $E(\varphi_1 U (\varphi_2 U \varphi_3))$, $E(\varphi_1 U (\varphi_2 \tilde{U} \varphi_3))$,
$E((X\varphi_1) \tilde{U} \varphi_2)$, $E((\varphi_1 \tilde{U} \varphi_2) \tilde{U} \varphi_3)$, $E((\varphi_1 U \varphi_2) \tilde{U} \varphi_3)$,
$E(\varphi_1 \tilde{U} (X\varphi_2))$, $E(\varphi_1 \tilde{U} (\varphi_2 \tilde{U} \varphi_3))$, or $E(\varphi_1 \tilde{U} (\varphi_2 U \varphi_3))$.

We assume that CTL^2 formulas are given in this form. We say that a CTL^2 formula φ is a *sad formula* iff it is of the form $E\varphi_1 \tilde{U}(\varphi_2 U \varphi_3)$. Otherwise, it is a *happy formula.* We now present CTL equivalences to all the happy formulas. Having a CTL equivalent to a CTL^2 formula φ means that if the maximal formulas in φ (i.e., the greatest strict subformulas of φ) are in CTL, then so is φ. A formal proof of the equivalences can be found in [BG94].

Equivalences for connected temporal operators

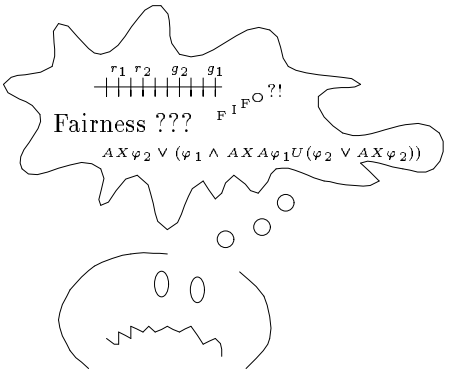
1. $E(\varphi_1 * (X\varphi_2)) \sim \varphi_1 * EX\varphi_2$
$E(\varphi_1 * (\varphi_2 U \varphi_3)) \sim \varphi_1 * E\varphi_2 U \varphi_3$

2. $E((X\varphi_1) * (X\varphi_2)) \sim EX(\varphi_1 * \varphi_2)$
$E((X\varphi_1) * (\varphi_2 U \varphi_3)) \sim (\varphi_3 * EX\varphi_1) \vee (\varphi_2 \wedge EX(\varphi_1 * E\varphi_2 U \varphi_3))$
$E((X\varphi_1) * (\varphi_2 \tilde{U} \varphi_3)) \sim ((\varphi_3 \wedge \varphi_2) * EX\varphi_1) \vee (\varphi_3 \wedge EX(\varphi_1 * E\varphi_2 \tilde{U} \varphi_3))$

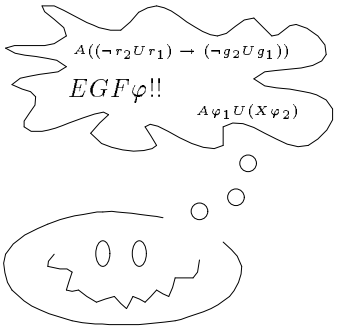
temporal operators in its path formula and the second formula indeed has only two temporal operators but they are neither nested nor connected by a binary Boolean operator in a simple way. It is easy to see that as far as syntax is concerned, CTL $\subset$ CTL$^2\subset$ CTL*.

4.1.2 Expressive Power of CTL^2

In this section we consider the expressive power of the logic CTL^2. The CTL* formula $AF(p \wedge Xp)$ which is not expressible in CTL^2 (this can be proved by an easy extension of Emerson and Halpern's proof from [EH86] for inexpressibility of $AF(p \wedge Xp)$ in CTL) testifies that CTL* is strictly more expressive than CTL^2. The CTL^2 formula $EGFp$, which is not expressible in CTL [EH86], testifies that CTL^2 is strictly more expressive than CTL. So, CTL $<$ CTL^2 $<$ CTL*.



1. A designer that uses CTL.



2. A designer that uses CTL^2.

We focus on the relation between CTL^2 and CTL. Definitely, an important advantage of CTL^2 over CTL is the increase in the expressive power. Consider the CTL^2 formula $A(\varphi_1 U(\varphi_2 \tilde{U} \varphi_3))$ which is not expressible in CTL. The formula is a base for many specifications used for reasoning about concurrent programs. Taking for instance $\varphi_1 = req_1$, $\varphi_2 = req_2$, and $\varphi_3 = CS_1$, it specifies the property "in all computation paths, process 1 keeps signaling req_1 until it eventually enters the critical section and stays there as long as process 2 does not signal req_2". Not less important, however, is the neat presentation that CTL^2 suggests for specifications whose CTL equivalences are very hard to understand. Indeed, the branching nature of CTL not only makes it difficult to understand a specification, but also makes it difficult for specifiers to express correctly their intuition. For example, the CTL^2 formula $A((\varphi_1 U \varphi_2) \rightarrow (\varphi_3 U \varphi_4))$ can express elegantly a FIFO policy. Taking $\varphi_1 = \neg req_1$, $\varphi_2 = req_2$, $\varphi_3 = \neg grant_1$, and $\varphi_4 = grant_2$, gives "in all computation paths, if req_1 does not precede req_2, then $grant_1$ does not precede $grant_2$". The CTL^2 formula $A(\varphi_1 \tilde{U}(X\varphi_2))$ enables specifying more accurate properties than these enabled without nesting of the next operator. Taking

problem and has been an active area of research [Eme90]. In this section, based on [BG94], we introduce such a language.

4.1.1 The Temporal Logic CTL^2

The logic CTL^2, introduced here, is a result of a new approach for defining sub-languages of CTL*. The syntax of CTL* allows path quantifiers to be applied to path formulas composed of an arbitrary combination of linear-time operators. The syntax of CTL restricts path quantifiers to be applied to path formulas with a single linear-time operator. The idea behind our approach is to allow a bounded number of linear-time operators within the path formulas of CTL*. In CTL^2, we investigate the simplest version of this extension: path formulas may contain an assertion composed of two, possibly negated, linear-time operators, either nested or connected by a binary Boolean operator. Formally, we distinguish between three types of CTL^2 formulas: state formulas, path formulas of degree 1, and path formulas of degree 2. Given a set AP of atomic propositions, a CTL^2 state formula is either:

S1. ***true***, ***false***, or p, for all $p \in AP$.

S2. $\neg\varphi_1$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are CTL^2 state formulas.

S3. $E\varphi_1$, where φ_1 is a CTL^2 path formula of degree 1 or 2.

A CTL^2 path formula of degree 1 is either:

P1. $X\varphi_1$, $\varphi_1 U \varphi_2$, or $\varphi_1 \tilde{U} \varphi_2$, where φ_1 and φ_2 are CTL^2 state formulas.

A CTL^2 path formula of degree 2 is either:

P2. $\varphi_1 \vee \varphi_2$ or $\varphi_1 \wedge \varphi_2$, where φ_1 is either a CTL^2 state formula or a CTL^2 path formula of degree 1, and φ_2 is a CTL^2 path formula of degree 1.

P3. $X\varphi_1$, $\varphi_1 U \varphi_2$, $\varphi_2 U \varphi_1$, $\varphi_1 \tilde{U} \varphi_2$, or $\varphi_2 \tilde{U} \varphi_1$, where φ_1 is a CTL^2 path formula of degree 1 and φ_2 is a CTL^2 state formula.

The logic CTL^2 consists of the set of state formulas generated by the above rules, where we also allow the usual abbreviations. Thus, formulas like $AGF\mathit{grant}$, $E(X\mathit{req})U\mathit{grant}$, $A(F\neg\mathit{busy} \vee G\mathit{wait})$, and $E(\mathit{busy}\tilde{U}\mathit{req})U\mathit{grant}$, are all CTL^2 formulas. Indeed, CTL^2 is strong enough to express neatly both liveness and safety properties used for reasoning about concurrent programs, such as unconditional fairness, preservation of FIFO order, more accurate specification of accessibility, absence of unsolicited response, and others. On the other hand, the formulas $A(GF\ \mathit{req} \rightarrow GF\ \mathit{grant})$ and $AF(\mathit{busy} \wedge X\mathit{busy})$ are not in CTL^2. The first formula has four

Chapter 4

Augmenting Branching Temporal Logics

Improving model-checking complexity is only one way to make this verification method practical. Naturally, there is a trade-off between the *expressive power* of a temporal logic and the *complexity* of its model-checking problem: the more a temporal logic is expressive, the higher its model-checking complexity is. A wise model checker should enable the maximal expressive power and ease of use within the complexity limits of the user.

In this chapter we consider three different ways of augmenting branching temporal logics. Each of the three ways makes branching temporal logic a more useful specification language. In Section 4.1 we show a simple yet powerful extension of the logic CTL. The extended logic, CTL^2, is strictly more expressive than CTL and is much more convenient as a specification language. Nevertheless, model checking of CTL^2 can be done with the same tools and is of the same complexity as is the one for the logic CTL. In Section 4.2 we consider the augmentation of branching temporal logics with existential quantification over atomic propositions. The resulted logics are very strong. Their computational price, however, is high. Quantification over atomic propositions has a lot in common with temporal operators that refer to the past. In Section 4.3 we study the augmentation of branching temporal logics with such operators.

4.1 Buy One, Get One Free

The exponential gap between CTL and LTL model-checking complexity led to a development of model checking tools for CTL [Bro86, McM93], while model checkers for LTL have lagged behind. Users of these tools, however, have to struggle with the limited expressive power of CTL. Unfortunately, the branching nature of CTL prevent them from checking many important behaviors. As a matter of course, finding specification languages which are strictly more expressive than CTL and yet maintain its attractive model-checking complexity is a challenging

assumption in the assume-guarantee pair concerns the interaction of the module with its environment along each computation, and is therefore more naturally expressed in a linear temporal logic. We denote this kind of assertion by $[\varphi]M\langle\psi\rangle$. The meaning of such an assertion is that the branching temporal formula ψ holds in the computation tree that consists of all computations of the program that satisfy the linear temporal formula φ. Verifying such assertions is called the *linear-branching modular model-checking problem*. This problem is studied in [Var95]. We now show that the branching modular framework is more general than the linear-branching modular framework.

Theorem 3.40 *For every LTL formula φ, module M, and a $\forall CTL^\star$ formula ψ, we have that $\langle A\varphi\rangle M\langle\psi\rangle$ iff $[\varphi]M\langle\psi\rangle$.*

Proof: To prove the claim, we need to generalize the notion of module. An *extended module* is a pair $\langle M, P\rangle$, where M is a module and P is a set of computations of M that start from some initial state. Formulas of $\forall$CTL* can be interpreted over an extended module $\langle M, P\rangle$ by letting path quantifiers range over suffixes of computations in P. The notion of simulation can be easily extended to extended modules. As detailed in [Var95], the truth of $[\varphi]M\langle\psi\rangle$ is determined by interpreting ψ over the extended module $\langle M_t, P_\varphi\rangle$, where M_t is the computation tree of M and P_φ is the set of computations of M_t that satisfy φ.

Given φ, M, and ψ, assume first that $[\varphi]M\langle\psi\rangle$ holds. Let M' be such that $M\|M' \models_f A\varphi$. Then, all fair computations in $M\|M'$ satisfy φ. Therefore, $M\|M' \leq \langle M_t, P_\varphi\rangle$, and $M\|M' \models_f \psi$. Thus, $\langle A\varphi\rangle M\langle\psi\rangle$.

Assume now that $\langle A\varphi\rangle M\langle\psi\rangle$ holds. Since $M_{A\varphi} \models_f A\varphi$, it follows that $M\|M_{A\varphi} \models_f \psi$. For every module M, we have that $\langle M_t, P_\varphi\rangle \leq M$ and $\langle M_t, P_\varphi\rangle \leq M_{A\varphi}$. Since $\langle M_t, P_\varphi\rangle \leq M$, we have that $\langle M_t, P_\varphi\rangle \leq \langle M_t, P_\varphi\rangle\|M$ too. Since $\langle M_t, P_\varphi\rangle \leq M_{A\varphi}$, we have that $\langle M_t, P_\varphi\rangle\|M \leq M_{A\varphi}\|M$ too. Thus, together, we have that $\langle M_t, P_\varphi\rangle \leq M\|M_{A\varphi}$. It follows that $\langle M_t, P_\varphi\rangle \models_f \psi$, which means that $[\varphi]M\langle\psi\rangle$ holds. □

It is known that the $\forall$CTL formula $AFAGp$ is not equivalent to any formula of the form $A\varphi$, where φ is an LTL formula [CD88].

Thus, the branching modular framework is strictly more expressive than the linear-branching modular framework. It is shown in [Var95] that linear-branching modular model checking is EXPSPACE-complete. Thus, the increase in expressive power is obtained with no increase in worst-case computational complexity. We note, however, that the algorithm in [Var95] for linear-branching modular model checking is simpler than the algorithm here, as it avoids the use of Safra's co-determinization construction.

Assume now that $M\|M_\varphi \models_f \psi$ and let $M\|M'$ be such that $M\|M' \models_f \varphi$. Then, $M\|M' \leq M_\varphi$, which implies that $M\|M\|M' \leq M\|M_\varphi$. Thus, $M\|M' \leq M\|M_\varphi$ and therefore $M\|M' \models_f \psi$. Hence, $\langle\varphi\rangle M\langle\psi\rangle$. □

Theorem 3.38 *Given a module M and two universal branching temporal logic formulas φ and ψ, the problem of determining whether $\langle\varphi\rangle M\langle\psi\rangle$ is solvable in space polynomial in $m\log(nl\cdot|M|)$ where m and n are the depth and the size, respectively, of an HAA corresponding to ψ, and l is the size of a maximal model for φ.*

Proof: By Theorem 3.37, the problem of determining whether $\langle\varphi\rangle M\langle\psi\rangle$ is reducible to model checking of ψ in $M\|M_\varphi$. The result then follows from Theorem 3.28. □

In Theorem 3.39 below we apply the method to the logics CTL and CTL*. We also show that the upper bounds that follow are tight.

Theorem 3.39

(1) *The branching modular model-checking problem for $\forall$CTL is PSPACE-complete.*

(2) *The branching modular model-checking problem for $\forall$CTL* is EXPSPACE-complete.*

Proof: The upper bounds follow from Theorem 3.38. The lower bound for $\forall$CTL* follows from [Var95] and the relation between the branching modular framework and the linear-branching modular framework (to be discussed in the next section). To get the lower bound for $\forall$CTL, we reduce the implication problem for $\forall$CTL to the branching modular model-checking problem for it. For a set AP of atomic propositions, let M_{AP} be the maximal model over AP. That is, $M_{AP} = \langle AP, 2^{AP}, 2^{AP}\times 2^{AP}, 2^{AP}, L, \{\langle 2^{AP}, \emptyset\rangle\}\rangle$ where for all $w \in 2^{AP}$ we have that $L(w) = w$. Let φ and ψ be $\forall$CTL formulas over a set AP of atomic propositions. Clearly, for every module M and $\forall$CTL formula φ over AP we have that $M\|M_{AP} \models_f \varphi$ iff $M \models_f \varphi$. Thus, the implication $\varphi \rightarrow \psi$ is valid iff $\langle\varphi\rangle M_{AP}\langle\psi\rangle$. The complexity of the reduction depends on the size of M_{AP}. Since the satisfiability problem for LTL is PSPACE-hard already for formulas with a fixed number of atomic propositions [SC85], so does the implication problem for $\forall$CTL. Thus, the size of M_{AP} is fixed. □

The Linear-Branching Modular Model Checking Problem

The modular proof rule in the preceding section uses branching assumptions and guarantees. As mentioned in the introduction, there is another approach in which the assumption is a linear temporal formula, while the guarantee is a branching temporal formula. In this approach, the

in $S_{E\varphi_1 U \varphi_2}$ that do not satisfy φ_2 in their initial state are required to satisfy φ_2 (rather than the less-economical $E\varphi_1 U \varphi_2$) in a successor state. Following the same idea, since the initial state of each module that satisfies $E\varphi_1 \tilde{U} \varphi_2$ must satisfy φ_2, the set of initial states of modules in $S_{E\varphi_1 \tilde{U} \varphi_2}$ is equal to the set of initial states of modules in S_{φ_2}. In addition, all the modules in $S_{E\varphi_1 \tilde{U} \varphi_2}$ satisfy $EG\varphi_2$, which is the most economical way to satisfy $E\varphi_1 \tilde{U} \varphi_2$. It is easy to prove, by an induction on the structure of φ, that each module in S_φ has at most $|\varphi|$ states and $|\varphi|$ transition and that if φ is satisfiable, then $S_\varphi \neq \emptyset$.

It is left to prove PSPACE completeness for the validity problem for $\forall$CTL*. Since a $\forall$CTL* formula ψ is valid iff the implication $\boldsymbol{true} \rightarrow \psi$ is valid, and since the implication problem $\varphi \rightarrow \psi$ for φ in $\forall$CTL and ψ in $\forall$CTL* is still in PSPACE, membership in PSPACE is easy. To prove hardness, we use the PSPACE-hardness of the validity problem for LTL. Clearly, an LTL formula ξ is valid iff the $\forall$CTL* formula $A\xi$ is valid.

□

3.6.4 Modular Model Checking

Branching Modular Model Checking

In branching modular verification, one uses assertions of the form $\langle\varphi\rangle M \langle\psi\rangle$ to specify that whenever M is part of a system satisfying the universal branching temporal logic formula φ, the system satisfies the universal branching temporal logic formula ψ too. Formally, $\langle\varphi\rangle M \langle\psi\rangle$ holds if $M \| M' \models_f \psi$ for all M' such that $M \| M' \models_f \varphi$. Here φ is an assumption on the behavior of the system and ψ is the guarantee on the behavior of the module. Assume-guarantee assertions are used in modular proof rules of the following form:

$$\left.\begin{array}{l} \langle \boldsymbol{true} \rangle M_1 \langle \varphi_1 \rangle \\ \langle \varphi_1 \rangle M_2 \langle \varphi_2 \rangle \\ \langle \varphi_2 \rangle M_1 \langle \varphi_3 \rangle \end{array}\right\} \langle \boldsymbol{true} \rangle M_1 \| M_2 \langle \varphi_3 \rangle$$

Thus, a key step in modular verification is checking that assume-guarantee assertions hold, which we called the *branching modular model-checking problem*. We now show that the techniques developed in the previous sections, maximal models and space-efficient fair model checking, yield also a solution to the branching modular model-checking problem.

Theorem 3.37 *For all $\forall$CTL* formulas φ and ψ, and for every module M, we have that $\langle\varphi\rangle M \langle\psi\rangle$ iff $M \| M_\varphi \models_f \psi$.*

Proof: Assume first that $\langle\varphi\rangle M \langle\psi\rangle$. Thus, whenever M is part of a system satisfying φ, the system satisfies ψ too. Since $M_\varphi \models_f \varphi$ and $M \| M_\varphi \leq M_\varphi$, we have that $M \| M_\varphi$ satisfies φ. Consequently, $M \| M_\varphi$ satisfies ψ.

For the upper bound, given a $\forall$CTL* formula φ, it is easy to see that φ is satisfiable iff the LTL formula obtained from φ by eliminating its path quantifiers is satisfiable.

Since $\forall$CTL subsumes propositional logic, hardness in NP for the validity problem of $\forall$CTL is immediate. To prove membership in NP, we prove that the satisfiability problem for $\exists$CTL is in NP. To do this, we present a linear-size model property for $\exists$CTL. Precisely, we prove that a satisfiable $\exists$CTL formula φ is satisfiable also in a module with at most $|\varphi|$ states and $|\varphi|$ transitions. Recall that each $\exists$CTL formula has one of the following forms: $\boldsymbol{true}$, $\boldsymbol{false}$, $p \in AP$, $\neg p$ for $p \in AP$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $EX\varphi_1$, $E\varphi_1 U \varphi_2$, or $E\varphi_1 \tilde{U} \varphi_2$, where φ_1 and φ_2 are $\exists$CTL formulas. With each $\exists$CTL formula φ, we associate a set S_φ of modules that satisfy φ. The definition of S_φ proceeds by induction on the structure of φ. Each module in S_φ has a single initial state. We use $S_{\varphi_1} \bowtie S_{\varphi_2}$ to denote the set of all modules obtained by taking a module M_1 from S_{φ_1} and a module M_2 from S_{φ_2}, such that M_1 and M_2 agree on the labeling of their initial states, and by unifying their initial states to a single initial state. We use $S_{\varphi_1} \rightarrow S_{\varphi_2}$ to denote the set of all modules obtained by taking a module M_1 from S_{φ_1}, a module M_2 from S_{φ_2}, adding a transition from the initial state of M_1 to the initial state of M_2, and fixing the initial state to be the one of M_1. We use $S_{\varphi_2} \downarrow$ to denote the set of all modules obtained by taking a module from S_{φ_2} and adding a self loop to its initial state.

- $S_{\boldsymbol{true}}$ is the set all one-state modules over AP.
- $S_{\boldsymbol{false}} = \emptyset$.
- S_p for $p \in AP$ is the set of all one-state modules over AP in which p holds.
- $S_{\neg p}$ for $\neg p \in AP$ is the set of all one-state modules over AP in which p does not hold.
- $S_{\varphi_1 \vee \varphi_2} = S_{\varphi_1} \cup S_{\varphi_2}$.
- $S_{\varphi_1 \wedge \varphi_2} = S_{\varphi_1} \bowtie S_{\varphi_2}$.
- $S_{EX\varphi_1} = S_{\boldsymbol{true}} \rightarrow S_{\varphi_1}$.
- $S_{E\varphi_1 U \varphi_2} = S_{\varphi_2} \cup (S_{\varphi_1} \rightarrow S_{\varphi_2})$.
- $S_{E\varphi_1 \tilde{U} \varphi_2} = S_{\varphi_2} \downarrow$.

Note that the definition of S_φ guarantees that for every module M which satisfies φ, there exists a module $M' \in S_\varphi$ such that M and M' agree on the labeling of their initial states. On the other hand, the modules in S_φ are economical with respect to states that are required for satisfaction of formulas that refer to the strict future. For example, since the initial state of each module that satisfy $E\varphi_1 U \varphi_2$ must satisfy either φ_1 or φ_2, the set of initial states of modules in $S_{E\varphi_1 U \varphi_2}$ is equal to the set of initial states of modules in S_{φ_1} or in S_{φ_2}. In addition, modules

The complexity of our method with respect to two logics L_1 and L_2 depends on the ability to associate with formulas of L_2 HAAs, in the size of these HAAs, their depth, as well as in the size of the maximal models for formulas in the logic L_1. In Theorem 3.35 below we apply the method to the logics CTL and CTL*. We also show that the upper bounds that follow are tight.

Theorem 3.35

(1) The implication problem for $\forall$CTL is PSPACE-complete.

(2) The implication problem for $\forall$CTL* is EXPSPACE-complete.

Proof: Theorems 3.32 and 3.33 provide us with a double exponential bound for maximal models of CTL* and an exponential bound for maximal models of CTL. Theorems 3.12 and 3.14 provide us with sizes and depth for HAAs of CTL and CTL*. Together with Theorem 3.34, this yields the upper bounds. To prove the PSPACE lower bound for the implication problem for $\forall$CTL, we prove that the satisfiability problem for $\forall$CTL is PSPACE-hard. The result then follows from Lemma 3.30. We prove hardness in PSPACE by a reduction from LTL satisfiability, proved to be PSPACE-hard in [SC85]. Given an LTL formula ξ, let ξ_A be the $\forall$CTL formula obtained from ξ by preceding each temporal operator with the path quantifier A. For example, if $\xi = FXp$ then $\xi_A = AFAXp$. It is easy to see that ξ is satisfiable iff ξ_A is satisfiable. Indeed, a computation that satisfies ξ can be viewed as a module satisfying ξ_A. Also, if ξ_A is satisfiable then it is also satisfiable in a module with a single computation. Clearly, this computation satisfies ξ. The EXPSPACE lower bound for the implication problem for $\forall$CTL* is similar to the EXPSPACE lower bound proved in [Var95] for linear-branching modular model checking. □

Note that the implication problem $\varphi \rightarrow \psi$ for φ in $\forall$CTL and ψ in $\forall$CTL* is still in PSPACE.

Recall that satisfiability and validity are special cases of the implication problem. As Theorem 3.36 below shows, these problems are easy special cases.

Theorem 3.36

(1) The satisfiability problems for $\forall$CTL and $\forall$CTL* are PSPACE-complete.

(2) The validity problem for $\forall$CTL is NP-complete.

(3) The validity problem for $\forall$CTL* is PSPACE-complete.

Proof: The results for the satisfiability follow from the PSPACE-completeness of the satisfiability problem for LTL. We have already seen the lower bound in the proof of the theorem 3.35.

that $\langle w, q\rangle \models_f \psi$. The proof proceeds easily by induction on the structure of ψ. In particular, satisfaction of formulas of the form $A\xi$ follows from the product with $\bar{\mathcal{A}}_{\forall(\varphi)}$.

Consider now a module $M = \langle AP, W_M, R_M, W_M^0, L_M, \alpha_M\rangle$ and assume that $M \models_f \varphi$. We show a simulation H from M to M_φ. For every state $w \in W_M$, define $f(w)$ to be the set of state formulas in $c(\varphi)$ which are true in w. The simulation H is defined as follows:

- For every $w \in W_M^0$, we have $H(w, \langle f(w), q_0\rangle)$.
- For every w_1, w_2 in W_M and $\langle f(w_1), q_1\rangle \in c(\varphi) \times Q$ such that $\langle w_1, w_2\rangle \in R_M$ and $H(w_1, \langle f(w_1), q_1\rangle)$, we have $H(w_2, \langle f(w_2), \delta(q_1, f(w_1))\rangle)$.

We prove that H is indeed a simulation from M to M_φ. That is, we prove that for all $w \in W_M^0$, there exists $w' \in W^0 \times \{q^0\}$ such that H is a simulation relation from $\langle M, w\rangle$ to $\langle M_\varphi, w'\rangle$. Consider a state $w \in W_M^0$. Clearly, $\langle f(w), q_0\rangle \in W^0 \times \{q_0\}$ and $H(w, \langle f(w), q_0\rangle)$. Now, let $w \in W_M$ and $\langle f(w), q\rangle \in c(\varphi) \times Q$ be such that $H(w, \langle f(w), q\rangle)$. By the definition of H, all the pairs in H are of this form. By the definition of L', we have that $L'(\langle f(w), q\rangle) = L_M(w)$. So, the first requirement on pairs in a simulation holds. For the second requirement, let $\pi = w_0, w_1, \ldots$ be a fair computation in M with $w_0 = w$. Consider the computation $\pi' = \langle f(w), q\rangle, \langle f(w_1), q_1\rangle, \ldots$ where $q_1 = \delta(q, f(w))$ and for every $i \geq 1$, we have $q_{i+1} = \delta(q_i, f(w_i))$. By the definition of H, we have that $H(w, \langle f(w), q\rangle)$ and for all $i \geq 1$ we have $H(w_i, \langle f(w_i), q_i\rangle)$. So, it only remains to show that π' is fair in M_φ. Since $M \models \varphi$ and π is fair, then for each state w_i and formula $A\xi \in \forall(\varphi)$ such that $A\xi \in f(w_i)$, we have that ξ is satisfied in $q_i, q_{i+1}, \ldots$. Thus, $q_i, q_{i+1}, \ldots$ is accepted by $\bar{\mathcal{A}}_{\forall(\varphi)}$ with q_i as an initial state and therefore, by the definition of β', the computation π' is fair. □

Theorem 3.33 *For every $\forall$CTL formula φ, there exists M_φ of size $2^{O(|\varphi|)}$.*

Proof: Exactly as for $\forall$CTL*. Here, however, $\bar{\mathcal{A}}_{\forall(\varphi)}$ is of size $2^{O(|\varphi|)}$. □

The Complexity of the Implication Problem

We are now ready to combine the maximal-model technique with the space-efficient fair-model-checking algorithm to solve the implication problem.

Theorem 3.34 *Given two universal branching temporal logic formulas φ and ψ, the implication problem $\varphi \rightarrow \psi$ can be solved in space polynomial in $m \log(nl)$ where m and n are the depth and the size, respectively, of an HAA corresponding to ψ, and l is the size of a maximal model for φ.*

Proof: By Theorem 3.31, the validity problem of $\varphi \rightarrow \psi$ is reducible to fair model checking of ψ in M_φ. The result then follows from Theorem 3.28. □

Büchi word automaton over $\Sigma = 2^{sf(\varphi)}$ such that $\mathcal{A}_{\forall(\varphi)}$ accepts an infinite word $\pi = w_0, w_1, \ldots$ iff there exists a suffix $w_i, w_{i+1}, \ldots$ of π and a formula $A\xi \in \forall(\varphi)$ such that $A\xi \in w_i$ and $w_i, w_{i+1}, \ldots$ does not satisfy ξ. That is, $\mathcal{A}_{\forall(\varphi)}$ nondeterministically guesses a location i and a formula $A\xi$ and then proceeds as the Büchi word automaton of $\neg\xi$. By [VW94], such $\mathcal{A}_{\forall(\varphi)}$ of size $2^{O(|\varphi|)}$ exists. Note that though ξ is a path formula of a branching temporal logic, we interpret it here over linear sequences. Since these sequences are labeled with all the branching subformulas of ξ, this causes no difficulty, as we can regard the branching subformulas of ξ as atomic propositions and regard ξ as a linear temporal logic formula.

We now take $\mathcal{A}_{\forall(\varphi)}$ and co-determinize it. The resulted automaton, called $\bar{\mathcal{A}}_{\forall(\varphi)}$, is a deterministic Rabin automaton which accepts exactly all the words $\pi = w_0, w_1, \ldots$ for which if a state w_i is labeled with some $A\xi \in \forall(\varphi)$, then ξ is satisfied in the suffix $w_i, w_{i+1}, \ldots$ of π. By [Saf89], the automaton $\bar{\mathcal{A}}_{\forall(\varphi)}$ is of size $2^{2^{O(|\varphi|)}}$.

For a set $s \subseteq sf(\varphi)$, we say that s is *consistent* iff the following four conditions hold:

1. For every $p \in AP$, if $p \in s$, then $\neg p \notin s$.
2. For every $p \in AP$, if $\neg p \in s$, then $p \notin s$.
3. For every $\varphi_1 \wedge \varphi_2 \in s$, we have that $\varphi_1 \in s$ and $\varphi_2 \in s$.
4. For every $\varphi_1 \vee \varphi_2 \in s$, we have that $\varphi_1 \in s$ or $\varphi_2 \in s$.

Let $c(\varphi)$ denote the set of all consistent subsets of $sf(\varphi)$. Consider the module

$$M = \langle AP, c(\varphi), c(\varphi) \times c(\varphi), W^0, L, \{\langle c(\varphi), \emptyset\rangle\}\rangle,$$

where the initial set W^0 includes all states $w \in c(\varphi)$ for which $\varphi \in w$, and for every $w \in c(\varphi)$, we have that $L(w) = w \cap AP$. That is, M is more general than any model of φ, yet, it is not necessarily a model of φ. To make it a maximal model, we take the product of M with $\bar{\mathcal{A}}_{\forall(\varphi)}$ as follows. Let $\bar{\mathcal{A}}_{\forall(\varphi)} = \langle \Sigma, Q, \delta, q^0, \beta\rangle$, where $\beta = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$. Then, $M_\varphi = \langle AP, c(\varphi) \times Q, R, W^0 \times \{q^0\}, L', \beta'\rangle$, where R, L', and β' are defined as follows.

- $R = \{\langle\langle w, q\rangle, \langle w', q'\rangle\rangle : \delta(q, w) = q'\}$.
- For all $w \in c(\varphi)$ and $q \in Q$, we have $L'(\langle w, q\rangle) = L(w)$.
- $\beta' = \{\langle c(\varphi) \times G_1, c(\varphi) \times B_1\rangle, \ldots, \langle c(\varphi) \times G_k, c(\varphi) \times B_k\rangle\}$.

We now prove the correctness of our construction. That is, we show that $M_\varphi \models_f \varphi$ and that for all modules M, we have $M \models_f \varphi$ only if $M \leq M_\varphi$ We first prove that $M_\varphi \models_f \varphi$. More precisely, we prove that for every state $\langle w, q\rangle \in c(\varphi) \times Q$, and for every formula $\psi \in w$, we have

3.6.3 Satisfiability, Validity, and Implication

Three basic decision problems for a specification language are *satisfiability*, i.e., is the specification trivially false, *validity*, i.e., is the specification trivially true, and *implication*, i.e., does one specification logically imply another specification. For a specification language that is closed under negation, these problems are inter-reducible. This is not the case, however, for universal branching temporal logics.

The satisfiability problem for a universal branching temporal logic is defined as follows: given a formula φ, is there a *nonempty* module M such that φ holds in M? As a module with no fair path trivially satisfies any $\forall$CTL* formula, the nonemptiness requirement is essential. The validity problem is defined as follows: given a formula φ, does φ hold in all modules? (It is easy to see that φ holds in all modules iff φ holds in all nonempty modules.) The implication problem is defined as follows: given two formulas φ and ψ, does ψ hold in every module in which φ holds? In Lemma 3.30 below, we show that the satisfiability and the validity problems are special cases of the implication problem.

Lemma 3.30 *Let φ be a formula in $\forall CTL^\star$. Then,*

(1) *The formula φ is satisfiable if and only if φ does not imply A**false**.*

(2) *The formula φ is valid if and only if **true** implies φ.*

Maximal Models and Implication

By definition, the implication $\varphi \rightarrow \psi$ is valid iff ψ holds in all modules M in which φ holds. Since the more behaviors M has the less likely it is to satisfy ψ, it makes sense to examine the implication by checking ψ in a maximal model of φ (we assume, without loss of generality, that φ and ψ are defined over the same set AP of atomic propositions). Formally, we have the following theorem. The theorem is proved in [GL94] for $\forall$CTL, but the proof applies also to $\forall$CTL*.

Theorem 3.31 *Let φ and ψ be $\forall CTL^\star$ formulas, and let M_φ be a maximal model of φ. Then φ implies ψ iff $M_\varphi \models_f \psi$.*

To complete the reduction of implication to fair model checking, we have to describe the construction of maximal models.

Theorem 3.32 *For every $\forall CTL^\star$ formula φ, there exists a maximal model M_φ of size $2^{2^{O(|\varphi|)}}$.*

Proof: For a $\forall$CTL* formula φ, let $sf(\varphi)$ denote the set of state subformulas of φ. Given φ, let $\forall(\varphi) \subseteq sf(\varphi)$ denote the set of all the state subformulas of φ of the form $A\xi$. Let $\mathcal{A}_{\forall(\varphi)}$ be a

- For $\beta_M = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$, $\beta = \{\langle G_1 \times Q_\psi, B_1 \times Q_\psi\rangle, \ldots, \langle G_k \times Q_\psi, B_k \times Q_\psi\rangle\}$.

Proposition 3.27

(1) $|\mathcal{A}_{M,\psi}| = O(|M| \cdot |\mathcal{A}_{\mathcal{D},\psi}|)$.

(2) $\mathcal{L}(\mathcal{A}_{M,\psi})$ *is nonempty if and only if* $M \models_f \psi$.

The proof of Proposition 3.27 is very similar to the proof of Proposition 3.2. The only difference is the fairness condition of the module, which is handled here by β. In particular, an infinite path ρ of a run r which gets trapped in the set $E\textbf{\textit{true}}$ should have $1 \leq i \leq k$ such that $Inf(r|\rho) \cap (G_i \times \{E\textbf{\textit{true}}\}) \neq \emptyset$ and $Inf(r|\rho) \cap (B_i \times \{E\textbf{\textit{true}}\}) = \emptyset$. Similarly, an infinite path ρ of a run r which gets trapped in the set $A\textbf{\textit{false}}$ should have $Inf(r|\rho) \cap (G_i \times \{A\textbf{\textit{false}}\}) = \emptyset$ or $Inf(r|\rho) \cap (B_i \times \{A\textbf{\textit{false}}\}) \neq \emptyset$ for all $1 \leq i \leq k$. As with the product of Kripke structures and HAAs, the depth of $\mathcal{A}_{M,\psi}$ is equal to the depth of $\mathcal{A}_{\mathcal{D},\psi}$.

In conclusion, given an HAA $\mathcal{A}_{\mathcal{D},\psi}$ such that $\mathcal{A}_{\mathcal{D},\psi}$ accepts exactly all the $\mathcal{D}$-trees that satisfy ψ, fair model checking of ψ in a module M with branching degrees in $\mathcal{D}$ is reducible to checking the 1-letter nonemptiness of a LAA of size $O(|M| \cdot |\mathcal{A}_{\mathcal{D},\psi}|)$ and of the same depth as $\mathcal{A}_{\mathcal{D},\psi}$. Thus, Theorem 3.26 implies the theorem below.

Theorem 3.28 *Given an HAA* $\mathcal{A}_{\mathcal{D},\psi}$ *of size* n *and of depth* m*, and a module* M *of size* l *with branching degrees in* $\mathcal{D}$*, fair model checking of* ψ *in* M *can be done using space* $O(m \log^2(nl))$.

Applying Theorem 3.28 to CTL and CTL*, we have the following:

Theorem 3.29

(1) The model-checking problem for CTL can be solved in space $O(m \log^2(mn))$*, where* m *is the length of the formula and* n *is the size of the Kripke structure.*

(2) The model-checking problem for CTL* *can be solved in space* $O(m(m + \log n)^2)$*, where* m *is the length of the formula and* n *is the size of the Kripke structure.*

In the following sections, we show how this result can be used to derive tight space complexity bounds also for the satisfiability, validity, and implication problems of universal branching temporal logics, as well as for the branching and the linear-branching modular model-checking problem.

transition from AXp, then not having p in the input letter still does not imply that we should reject; it might be that all the paths in the input tree are not fair.

Generally, when considering fair model checking, we attribute each atomic proposition and each negation of an atomic proposition with either A or E. As every state formula of the form $E\xi$, the formula Ep, for an atomic proposition p, means that there exists a fair path for which p holds in its first position. Thus, $w \models_f Ep$ iff $p \in L(w)$ and there exists a fair path starting at w. Similarly, $w \models_f Ap$ iff $p \in L(w)$ or there exists no fair path starting at w. Below we adjust HAAs to handle formulas with attributed atomic propositions. Each adjusted HAAs has two new states, $A\boldsymbol{false}$ and $E\boldsymbol{true}$, and the following new transitions:

- $\delta(Ap, \sigma, k) = \boldsymbol{true}$ if $p \in \sigma$.
- $\delta(Ap, \sigma, k) = A\boldsymbol{false}$ if $p \notin \sigma$.
- $\delta(A\neg p, \sigma, k) = \boldsymbol{true}$ if $p \notin \sigma$.
- $\delta(A\neg p, \sigma, k) = A\boldsymbol{false}$ if $p \in \sigma$.
- $\delta(A\boldsymbol{false}, \sigma, k) = \bigwedge_{c=0}^{k-1}(c, A\boldsymbol{false})$.
- $\delta(Ep, \sigma, k) = E\boldsymbol{true}$ if $p \in \sigma$.
- $\delta(Ep, \sigma, k) = \boldsymbol{false}$ if $p \notin \sigma$.
- $\delta(E\neg p, \sigma, k) = E\boldsymbol{true}$ if $p \notin \sigma$.
- $\delta(E\neg p, \sigma, k) = \boldsymbol{false}$ if $p \in \sigma$.
- $\delta(E\boldsymbol{true}, \sigma, k) = \bigvee_{c=0}^{k-1}(c, E\boldsymbol{true})$.

In addition, we adjust the acceptance condition to be $\langle G \cup \{E\boldsymbol{true}\}, B \cup \{A\boldsymbol{false}\}\rangle$, where $\langle G, B\rangle$ is the acceptance condition of the original HAA (namely, the one that corresponds to the same formula without the attribution of the atomic propositions). It is easy to see that the state $A\boldsymbol{false}$ constitutes a universal set and that the state $E\boldsymbol{true}$ constitutes an existential set. The task of the states $A\boldsymbol{false}$ and $E\boldsymbol{true}$ will get clearer after the definition of the product automaton. For now, note that the language of an adjusted HAA is exactly the same as the language of the original HAA. It is only that some finite paths in a run of the original HAA may now be infinite and get trapped in the existential set $E\boldsymbol{true}$ and that some runs of the adjusted HAA that have infinite paths which get trapped in the universal set $A\boldsymbol{false}$ are not possible runs of the original HAA.

Given a branching temporal logic formula ψ and a set $\mathcal{D} \subset \mathbb{N}$ of branching degrees, let $\mathcal{A}_{\mathcal{D},\psi} = \langle \Sigma, \mathcal{D}, Q_\psi, \delta_\psi, \psi, \alpha_\psi\rangle$ be an HAA which accepts exactly all the $\mathcal{D}$-trees that satisfy ψ. Consider a module $M = \langle \Sigma, W, R, W^0, L, \beta_M\rangle$ with branching degrees in $\mathcal{D}$. The product automaton of M and $\mathcal{A}_{\mathcal{D},\psi}$ is the LAA $\mathcal{A}_{M,\psi} = \langle \{a\}, \mathcal{D}, W \times Q_\psi, \delta, W^0 \times \{\psi\}, \alpha, \beta\rangle$, where δ, α, and β are defined as follows:

- Let $q \in Q_\psi$, $w \in W$, $succ_R(w) = \langle w_0, \ldots, w_{bd(w)-1}\rangle$, and $\delta_\psi(q, L(w), bd(w)) = \theta$. Then $\delta(\langle w, q\rangle, a, bd(w)) = \theta'$, where θ' is obtained from θ by replacing each atom (c, η) in θ by the atom $(c, \langle w_c, \eta\rangle)$. For $k \neq bd(w)$, we have $\delta(\langle w, q\rangle, a, k) = \boldsymbol{false}$.
- For $\alpha_\psi = \langle G, B\rangle$, we have that $\alpha = \langle W \times G, W \times B\rangle$.

1. $q' \in G$.
2. $q'' \in G_j$.
3. q' is reachable from q.
4. q'' is $Q_i \setminus B_j$-reachable from q'.
5. q' is $Q_i \setminus B_j$-reachable from q''.

If such q' and q'' are found (possibly $q' = q''$), label q with 'T'.

- For a universal set Q_i, we now have

  (4.2) If no such state exists, guess $1 \leq j \leq k$ and search for states q' and q'' of Q_i such that the following hold:

  1. $q' \in B$.
  2. $q'' \in G_j$.
  3. q' is reachable from q.
  4. q'' is $Q_i \setminus B_j$-reachable from q'.
  5. q' is $Q_i \setminus B_j$-reachable from q''.

  If such q' and q'' are found, label q with 'F'.

The idea of the algorithm is straightforward: in an existential set, finding q and q' as required witnesses a path that satisfies both α and β. In a universal set, finding q and q' as required witnesses a path that satisfies β and does not satisfy α. Since we only elaborate the reachability checks, the complexity of the algorithm is the same as in Theorem 3.17. □

Fair Model Checking

The method presented in Section 3.2 reduces branching-time model checking to 1-letter nonemptiness of alternating tree automata. We now extend this method and reduce fair branching-time model checking to 1-letter nonemptiness of LAAs. To do this, we define the product of a module and an alternating tree automaton, extending the definition of a product of a Kripke structure and an alternating tree automaton, described in Section 3.2.2. The alternating tree automata we consider here are HAAs. In this case, the resulted product automaton is a LAA.

Before defining the product, we have to adjust the HAAs to handle fairness. Consider the HAA of some CTL formula and consider a state p in it which is associated with an atomic proposition. We may reach the state p in a transition from, say, a state EXp. Then, when the HAA is in state p, it is not sufficient to have p in the input letter; we also have to make sure that the HAA can proceed along a fair path of the input tree. Similarly, if we reach p in a

(3) For every M, M', and M'', if $M \leq M'$ then $M \| M'' \leq M' \| M''$.

(4) For every M, we have $M \leq M \| M$.

(5) For every M and M' such that $M \leq M'$, and for every universal branching temporal logic formula φ, $M' \models_f \varphi$ implies $M \models_f \varphi$.

A module M is a *maximal model* for an $\forall$CTL* formula φ if it allows all behaviors consistent with φ. Formally, M is a maximal model of φ if $M \models_f \varphi$ and for every module M' we have that $M' \leq M$ if $M' \models_f \varphi$. Note that by the preceding theorem, if $M' \leq M$, then $M' \models_f \varphi$. Thus, M_φ is a maximal model for φ if for every module M, we have that $M \leq M_\varphi$ iff $M \models \varphi$. A construction of maximal models for $\forall$CTL formulas is described in [GL94]. We will describe later a construction of maximal models for $\forall$CTL* formulas.

3.6.2 An Automata-Theoretic Framework to Fair Model Checking

Libi Alternating Automata

In order to perform fair model checking, we introduce *Libi alternating automata* (LAAs). A LAA $\mathcal{A} = \langle \Sigma, Q, \delta, Q^0, \alpha, \beta \rangle$ is an HAA extended with a Rabin acceptance condition $\beta \subseteq 2^Q \times 2^Q$. For a run r of a LAA with $\alpha = \langle G, B \rangle$ and $\beta = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$, a path ρ which gets trapped within a set Q_i is accepted by the run iff either Q_i is an existential set, in which case $Inf(r|\rho) \cap G \neq \emptyset$ and there exists $1 \leq j \leq k$ such that $Inf(r|\rho) \cap G_j \neq \emptyset$ and $Inf(r|\rho) \cap B_j = \emptyset$, or Q_i is a universal set, in which case $Inf(r|\rho) \cap B = \emptyset$ or, for all $1 \leq j \leq k$, either $Inf(r|\rho) \cap G_j = \emptyset$ or $Inf(r|\rho) \cap B_j \neq \emptyset$. This means that we require paths that get trapped in an existential set to satisfy both α and β, and that we require paths that get trapped in a universal set and satisfy β to satisfy α too.

We now show that the space complexity result presented in Theorem 3.17 for the 1-letter nonemptiness of HAAs extends to LAAs.

Theorem 3.26 *The 1-letter nonemptiness problem for a LAA of size n and with m sets can be solved in space $O(m \log^2 n)$.*

Proof: Consider a LAA with $\alpha = \langle G, B \rangle$ and $\beta = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$. We extend the nonemptiness procedure presented for HAAs in the proof of Theorem 3.17. The only change required is in steps (3.2) and (4.2) that handle existential and universal sets. For a set $S \subseteq Q$ and two states q and q', we say that q' is *S-reachable* from q, if q' is reachable from q using states from S only (in particular, q and q' are members of S).

- For an existential set Q_i, we now have

  (3.2) If no such state exists, guess $1 \leq j \leq k$ and search for states q' and q'' of Q_i such that the following hold:

(2.1) $L(s) \cap AP' = L(s')$.

(2.2) For every fair computation $\pi = s_0, s_1, \ldots$ in M, with $s_0 = s$, there exists a fair path $\pi' = s'_0, s'_1, \ldots$ in M', with $s'_0 = s'$, such that for all $i \geq 0$, we have $H(s_i, s'_i)$.

A simulation relation H is a *simulation from M to M'* iff for every $w \in W^0$ there exists $w' \in W^{0'}$ such that $H(w, w')$. If there exists a simulation from M to M', we say that M *simulates* M' and we write $M \leq M'$. Intuitively, it means that the module M' has more behaviors than the module M. In fact, every possible behavior of M is also a possible behavior of M'. Note that our simulation is different from the classical simulation used by Milner [Mil71], where there are no fairness conditions.

Let M and M' be two modules. The *composition* of M and M', denoted $M \| M'$, is a module hat has exactly these behaviors which are joint to M and M' and are fair in both of them. In order to define $M \| M'$, we define first the structure M'', which has no fairness condition and which has exactly the joint behaviors of M and M', ignoring their fairness conditions. Formally, $M'' = \langle AP'', W'', R'', W^{0''}, L'' \rangle$, where:

- $AP'' = AP \cup AP'$.
- $W'' = \{\langle w, w' \rangle : L(w) \cap AP' = L(w') \cap AP\}$.
- $R'' = \{\langle \langle w, w' \rangle, \langle s, s' \rangle \rangle : \langle w, s \rangle \in R \text{ and } \langle w', s' \rangle \in R'\}$.
- $W^{0''} = (W^0 \times W^{0'}) \cap W''$.
- For every $\langle w, w' \rangle \in W''$, we have $L''(\langle w, w' \rangle) = L(w) \cup L'(w')$.

In order to get $M \| M'$, we have to elaborate M'' such that its behaviors will be fair in both M and M'. Clearly, a computation π in M'' is fair iff there exist a pair $\langle G, B \rangle \in \alpha$ and a pair $\langle G', B' \rangle \in \alpha'$ such that π visits states from $G \times W'$ infinitely often, visits states from $W \times G'$ infinitely often, and visits states from $(B \times W') \cup (W \times B')$ only finitely often. Thus, checking the fairness of π with respect to a single pair of pairs can be done, as in the product of two Büchi automata, by augmenting each state of M'' with a Boolean flag. For mm' such pairs of pairs (originating from m pairs in α and m' pairs in α'), checking the fairness of π can be done, by augmenting each state of M'' with a $2mm'$-valued flag, resulting in $M \| M'$ with $2mm'|W''|$ states.

Theorem 3.25 *[GL94]*

(1) *The simulation relation $\leq$ is a preorder.*

(2) *For every M and M', we have $M \| M' \leq M$.*

A $\forall$CTL* path formula is either:

- A CTL* state formula.
- $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $X\psi_1$, $\psi_1 U \psi_2$, or $\psi_1 \tilde{U} \psi_2$, where ψ_1 and ψ_2 are $\forall$CTL* path formulas.

The logic $\forall$CTL* consists of the set of state formulas generated by the above rules. Note that since negation in $\forall$CTL* can be applied only to atomic propositions, assertions of the form $\neg A\psi$, which is equivalent to $E\neg\psi$, are not possible. The logic $\forall$CTL is defined similarly, as the restricted subset of CTL which allows the universal path quantifier only. The logics $\exists$CTL* and $\exists$CTL are defined analogously, as the existential fragments of CTL* and CTL, respectively. Note that negating a $\forall$CTL* formula results in an $\exists$CTL* formula.

We define the semantics of $\forall$CTL* with respect to *fair modules* (*modules*, for short). Modules augment Kripke structures with a fairness condition. In addition, a module may have several initial states. Precisely, a module $M = \langle AP, W, R, W^0, L, \alpha \rangle$ consists of a set AP of atomic propositions, a set W of states, a total transition relation $R \subseteq W \times W$, a set $W^0 \subseteq W$ of initial states, a labeling function $L : W \rightarrow 2^{AP}$, and a Rabin fairness condition α (our choice of this type of fairness condition is technically motivated, as will be clarified in the sequel). A computation of a module is a sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, we have that $\langle w_i, w_{i+1} \rangle \in R$. A computation of M is *fair* iff it satisfies the fairness condition α. That is, if $\alpha = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$ then π is fair iff there exists $1 \leq i \leq k$ such that $w_j \in G_i$ for infinitely many j's and $w_j \in B_i$ for only finitely many j's. When defining the semantics of $\forall$CTL* over modules, path quantifiers range only over fair computations, cf. [CES86]. A module M satisfies a formula φ iff φ holds in *all* initial states of M. We say that a module is *nonempty* iff there exists a fair computation that starts at an initial state. We use $M \models_f \varphi$ to denote that a module M satisfies the $\forall$CTL* formula φ.

Simulation Relation and Composition of Modules

In the context of modular verification, it is helpful to define an order relation between modules [GL94]. Intuitively, the order captures what it means for a module M' to have "more behaviors" than a module M.

Let $M = \langle AP, W, R, W^0, L, \alpha \rangle$ and $M' = \langle AP', W', R', W^{0'}, L', \alpha' \rangle$ be two modules for which $AP' \subseteq AP$, and let w and w' be states in W and W', respectively. A relation $H \subseteq W \times W'$ is a *simulation relation* from $\langle M, w \rangle$ to $\langle M', w' \rangle$ iff the following conditions hold:

(1) $H(w, w')$.

(2) For all s and s', we have that $H(s, s')$ implies the following:

are not closed under negation, these problems are not inter-reducible as they are for CTL and CTL*).

We use two fundamental techniques to solve these questions. The first technique is the *maximal-model* technique introduced in [GL94]. It is shown there that with every $\forall$CTL formula φ one can associate a *maximal model* M_φ (called the *tableau* of φ in [GL94]) such that a module M satisfies φ precisely when M *simulates* M_φ (we define simulation later on). We use here automata-theoretic techniques for CTL* [VS85, EJ88] to construct maximal models for $\forall$CTL* formulas. While maximal models for $\forall$CTL involve an exponential blow-up, maximal models for $\forall$CTL* involve a doubly exponential blow-up.

The second technique is the automata-theoretic framework to branching-time model checking introduced in Section 3.2. There, we showed how to use alternating tree automata to obtain space-efficient model checking methods. Since the maximal models that we construct include fairness conditions, we extend the automata-theoretic method to yield space-efficient *fair* model-checking algorithms. We then show how performing fair model checking over maximal models can solve the satisfiability, validity, and implication problems for $\forall$CTL and $\forall$CTL*. Our results show that these problems are computationally easier than the analogous problems for CTL and CTL*. For example, while all three problems are EXPTIME-complete for CTL, we have that satisfiability and implication are PSPACE-complete and validity is NP-complete for $\forall$CTL.

We turn on to the branching modular model-checking problem. Interestingly, the same two fundamental techniques of maximal models and space-efficient fair model checking also yield a solution to this problem. We prove that the branching modular model-checking problem is PSPACE-complete for $\forall$CTL and EXPSPACE-complete for $\forall$CTL*. We show that the increase in complexity is solely to the assumption part of the specification. This suggests that modular model checking in the branching temporal framework can be practical only for very small assumptions.

3.6.1 Preliminaries

Universal Branching Temporal Logics

The logic $\forall CTL^\star$ is a restricted subset of CTL* which allows only the universal path quantifier. Formally, let AP be a set of atomic proposition names. A $\forall$CTL* state formula is either:

- ***true***, ***false***, p, or $\neg p$, for all $p \in AP$.
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are $\forall$CTL* state formulas.
- $A\psi_1$, where ψ_1 is an $\forall$CTL* path formula.

computations of the module are guaranteed to satisfy ψ, assuming that all the computations of the environment satisfy φ. As observed in [Pnu85], in this case the assume-guarantee pair $\langle \varphi, \psi \rangle$ can be combined to a single linear temporal logic formula $\varphi \rightarrow \psi$. Thus, model checking a module with respect to assume-guarantee specifications in which both the assumed and the guaranteed behaviors are linear temporal logic formulas is essentially the same as model checking the module with respect to linear temporal logic formulas.

The situation is different for the branching temporal paradigm. Here the guarantee is a branching temporal formula, which describes the computation tree of the module. There are two approaches, however, to the assumptions in assume-guarantee pairs. The first approach, implicit in [CES86, EL85b, EL87] and made explicit in [Jos87a, Jos87b, Jos89, DDGJ89], is that the assumption in the assume-guarantee pair concerns the interaction of the module with its environment along each computation, and is therefore more naturally expressed in linear temporal logic. Thus, in this approach an assume-guarantee pair should consist of a *linear* temporal assumption φ and a *branching* temporal guarantee ψ. The meaning of such a pair is that ψ holds in the computation tree that consists of all computations of the program that satisfy φ. The problem of verifying that a given module M satisfies such a pair $\langle \varphi, \psi \rangle$, which we call the *linear-branching modular model-checking problem*, is more general than either linear or branching model checking and is studied in depth in [Var95].

A second approach was considered in [GL94], where assumptions are taken to apply to the computation tree of the system within which the module is interacting. Accordingly, assumptions in [GL94] are also expressed in branching temporal logic. There, a module M satisfies an assume-guarantee pair $\langle \varphi, \psi \rangle$ iff whenever M is part of a system satisfying φ, the system satisfies ψ too. We call this *branching modular model checking*. Furthermore, it is argued there, as well as in [DDGJ89, Jos89, GL91, DGG93], that in the context of modular verification it is advantageous to use only *universal* branching temporal logic, i.e., branching temporal logic without existential path quantifiers. That is, in universal branching temporal logic one can state properties of all computations of a program, but one cannot state that certain computations exist. Consequently, universal branching temporal logic formulas have the helpful property that once they are satisfied in a module, they are satisfied also in a system that contains this module. The focus in [GL94] is on using $\forall$CTL, the universal fragment of CTL, for both the assumption and the guarantee.

In this section, based on [KV95], we focus on the branching modular model-checking problem, which we show to be a proper extension of the linear-branching modular model-checking problem. That is, the linear-branching modular model-checking problem is a special case of the branching modular model-checking problem. We consider assumptions and guarantees in both $\forall$CTL and in the more expressive $\forall$CTL*. We start by examining the most fundamental questions about these logics: *satisfiability*, *validity*, and *implication* (since $\forall$CTL and $\forall$CTL*

of P is of size exponential in the size of P. According to [EL86], model checking of a Kripke structure K with respect to a μ-calculus formula ψ is of time complexity $O((|\psi| \cdot |K|)^{n+1})$, where n is the alternation depth of ψ. The alternation depth of a formula ψ is the maximal number of alternations between μ and ν on any syntactic path from an occurrence of μy or νy to an occurrence of y. A formal definition can be found in [EL86], yet the one we gave here is good enough to get convinced that $n < |\psi|$. Therefore, model checking of K with respect to ψ is of time complexity $O((|\psi| \cdot 2^{|P|})^{|\psi|})$. Hence, fixing ψ we get that the program complexity of μ-calculus model checking for concurrent programs is in EXPTIME. □

To conclude, the automata-theoretic framework provides improved (and significant) space-complexity upper bounds for the model checking problem of CTL and CTL* and explains why similar improved bounds can not be obtained for model checking of the μ-calculi.

3.6 The Complexity of Modular Model Checking

Even with the improved space complexity results, model checking still suffers from the state-explosion problem. Modular verification is one possible way to address the state-explosion problem [CLM89, GL94, ASSSV94]. In modular verification, we use proof rules of the following form:

$$\left.\begin{array}{l} M_1 \models \psi_1 \\ M_2 \models \psi_2 \\ C(\psi_1, \psi_2, \psi) \end{array}\right\} M_1 \| M_2 \models \psi$$

Here, $C(\psi_1, \psi_2, \psi)$ is some logical condition relating ψ_1, ψ_2, and ψ. The advantage of using modular proof rules is that it enables us to apply model checking to components of a system (which we call *modules*) and deduce the correctness of the entire system. Obviously, the modules of the system have much smaller state spaces.

The state-explosion problem is only one motivation for pursuing modular verification. Modular verification is advocated also for other methodological reasons; a robust verification methodology should provide rules for deducing properties of programs from the properties of their constituent modules. Indeed, efforts to develop modular verification frameworks were undertaken in the mid 1980s [Pnu85].

A key observation is that in modular verification the specification should include two parts [Lam83]. One part describes the desired behavior of the module. The other part describes the assumed behavior of the system within which the module is interacting. This is called the *assume-guarantee* paradigm, as the specification describes what behavior the module is *guaranteed* to exhibit, *assuming* that the system behaves in the promised way.

For the linear-time paradigm, an assume-guarantee specification is a pair $\langle \varphi, \psi \rangle$, where both φ and ψ are linear temporal logic formulas. The meaning of such a pair is that all the

corresponds to the formula $\psi = \mu y.(p \vee EXAXy)$ and in which an alternation between existential and universal states that belong to the same set is possible. So, while the syntax of CTL and CTL* makes the alternation of the path quantifiers A and E bounded by the length of the formula, the fixed-point operators in μ-calculus enable an unbounded alternation. This unbounded alternation is the key for the following theorem:

Theorem 3.23 *The program complexity of alternation-free μ-calculus model checking for concurrent programs is EXPTIME-complete.*

Proof: Clearly, the problem can be solved in EXPTIME by building the nondeterministic program corresponding to the concurrent program and using the model-checking algorithm from Section 3.3.1. To prove that it is hard in EXPTIME, we do a reduction from alternating linear-space Turing machines, proved to be EXPTIME-hard in [CKS81].

Similarly to what we have done proving Theorem 3.20, we show that there exists an alternation-free μ-calculus formula ψ such that given an alternating Turing machine T of space complexity $s(n)$ and a word w of length n, it is possible to build, with a logarithmic space construction, a concurrent program P of size $O(s(n))$ such that P satisfies ψ if and only if T accepts w. The model of alternation we use is that T is universal in its even steps and is existential in its odd steps.

Given T, the construction of P is exactly the same as in Theorem 3.20. Consider the μ-calculus formula

$$\psi = \mu y.(accept \vee EX(accept \vee AXy)).$$

$P \models \psi$ if and only if there exists a computation of T on w in which all the leaves of the computation tree eventually reach an accepting state. Thus, $P \models \psi$ if and only if T accepts the empty tape. □

Theorem 3.23 implies that while the time complexities of model checking for CTL and alternation-free μ-calculus coincide, there is a gap between the space complexities of model checking for these logics. Moreover, the program complexity of model checking for CTL* is lower than the program complexity of model checking for the alternation-free μ-calculus. In Theorem 3.24 we go further, showing that the program complexities of model checking for concurrent programs for alternation-free μ-calculus and general μ-calculus coincide.

Theorem 3.24 *The program complexity of μ-calculus model checking for concurrent programs is EXPTIME-complete.*

Proof: Clearly, hardness in EXPTIME follows from Theorem 3.23. To prove membership in EXPTIME we use the algorithm suggested in [EL86]. Given a concurrent program P and a μ-calculus formula ψ, the size of a Kripke structure K that models the nondeterministic expansion

Theorem 3.21 *The program complexity of CTL and CTL* model checking for concurrent programs is PSPACE-complete.*

Proof: Clearly, the upper bound proved in Theorem 3.20 holds here too. Since proving the lower bound we used a fixed formula, hardness in PSPACE holds too. □

The fact that CTL and CTL* formulas can be translated to HAAs plays a crucial role in our upper bound. To see this, we prove in Theorem 3.22 bellow, that the 1-letter nonemptiness problem for WAAs is P-complete. Thus, the restricted structure of HAAs is essential for a space-efficient nonemptiness test.

Theorem 3.22 *The 1-letter nonemptiness problem for weak alternating automata is P-complete.*

Proof: Membership in P follows from Theorem 3.9. Hardness in P follows by a reduction from the Alternating Graph Accessibility problem, proved to be P-complete in [Imm81, CKS81, GHR91], to nonemptiness of weak alternating word automata. The result for WAAs then follows by Theorem 3.1. In the Alternating Graph Accessibility problem, we are given a directed graph $G = \langle V, E\rangle$, a partition $\mathcal{E} \cup \mathcal{U}$ of V, and two designated vertices s and t. The problem is whether *alternating_path*(s,t) is true, where *alternating_path*(x,y) holds if and only if:

1. $x = y$, or
2. $x \in \mathcal{E}$ and there exists z with $\langle x,z\rangle \in E$ and *alternating_path*(z,y), or
3. $x \in \mathcal{U}$ and for all z with $\langle x,z\rangle \in E$, we have *alternating_path*(z,y).

Given $G, \mathcal{E}, U, s$, and t, we define the weak alternating word automaton $\mathcal{A} = \langle \{a\}, V, \delta, \{s\}, \{t\}\rangle$ where δ is defined as follows:

- If $q \in \mathcal{E}$ and $q \neq t$, then $\delta(q,a) = \bigvee_{\langle q,q'\rangle \in E} q'$.
- If $q \in \mathcal{U}$ and $q \neq t$, then $\delta(q,a) = \bigwedge_{\langle q,q'\rangle \in E} q'$.
- $\delta(t,a) = \boldsymbol{true}$.

It is easy to see that a partition of V into two sets, $V \setminus \{t\}$ and $\{t\}$, satisfies the weakness requirements and that the language of $\mathcal{A}$ is not empty iff *alternating_path*(s,t). □

We now consider the program complexity of μ-calculus and alternation-free μ-calculus model checking for concurrent programs. The WAAs that correspond to alternation-free μ-calculus do not have the restricted structure of HAAs. In Example 3.7, we presented the WAA that

of the Kripke structure. Since the product of the components of a concurrent program is at most exponentially larger than the program, membership in PSPACE is immediate. To prove hardness in PSPACE, we do a reduction from polynomial space Turing machines.

Precisely, we show that there exists a CTL formula ψ such that given a Turing machine T of space complexity $s(n)$ and a word w of length n, it is possible to build, with a logarithmic space construction, a concurrent program P of size $O(s(n))$ such that P satisfies ψ if and only if T accepts w.

Let $T = \langle \Gamma, Q, \rightarrow, q_0, F \rangle$ be a Turing machine where Γ is the alphabet, Q is the set of states, $\rightarrow \subseteq Q \times \Gamma \times Q \times \Gamma \times \{R, L\}$ is the transition relation (we use $(q, a) \rightarrow (q', b, \Delta)$ to indicate that when T is in state q and it reads the input a in the current tape cell, it moves to state q', writes b in the current tape cell, and its reading head moves one cell to the right/left, according to Δ), q_0 is the initial state, and $F \subseteq Q$ is the set of accepting states. Let $w = w_1 \cdots w_n$. The concurrent program P has $s(n)$ processes, one for each tape cell that is used. For all $1 \leq i \leq s(n)$, the process P_i is defined as follows.

1. $AP_i = \{accept_i\}$ and $AC_i = \{i-1, i, i+1\} \times Q$.
2. The state set of P_i is $(Q \times \Gamma) \cup \Gamma$. A state of the form (q, a) indicates that T is in state q, its reading head is on cell i, and the content of cell i is a. A state of the form a indicates that the content of cell i is a and the reading head is not on cell i.
3. For each transition $(q, a) \rightarrow (q', b, \Delta)$ of T, we have the following transitions in P_i.
   (a) A transition from (q, a) to b labeled by $(i+1, q')$ if $\Delta = R$ and by$(i-1, q')$ if $\Delta = L$. This transition corresponds to the head moving from cell i to cell $i+1$ or $i-1$.
   (b) A transition from every $a \in \Gamma$ to (q', a) labeled by (i, q'). This transition corresponds to the head moving to cell i from cell $i+1$ or $i-1$.
4. The initial state of P_i corresponds to the initial content of cell i. Thus, it is (q_0, w_1) for $i = 1$, it is w_1 for $1 < i \leq n$, and it is ϵ for $n < i \leq s(n)$.
5. We label a state (q, a) with $accept_i$ if and only if $q \in F$.

The concurrent behavior of the processes embodies all the computations of T on w. To see this, observe that each reachable state s in P has exactly one $1 \leq i \leq n$ for which $s[i] \in Q \times \Gamma$. Thus, each reachable state in P corresponds to a configuration of T. Also, a transition from a state s_1 to a state s_2 corresponds to a possible transition from the configuration associated with s_1 to the one associated with s_2.

Now, let $accept = \bigvee_i accept_i$. Consider the CTL formula $\psi = EF accept$. $P \models \psi$ if and only if there exists a computation of T on w which eventually reaches an accepting state. Thus, $P \models \psi$ if and only if T accepts w. $\square$

3.5 The Complexity of Model Checking for Concurrent Programs

We consider a concurrent program P composed of n concurrent processes P_i. Each process is described by a transition system $D_i = \langle AP_i, AC_i, S_i, \Delta_i, s_{0_i}, L_i \rangle$ where AP_i is a set of atomic propositions, AC_i is an action alphabet, S_i is a finite set of states, $\Delta_i \subseteq S_i \times AC_i \times S_i$ is a transition relation, $s_{0_i} \in S_i$ is an initial state, and $L_i : S_i \to 2^{AP_i}$ maps each state to the set of atomic propositions true is this state. We require that the atomic-proposition sets of the processes are disjoint.

A concurrent behavior of these processes is defined by the usual interleaving semantics: actions that appear in several processes are synchronized by common actions. Using this convention, one can obtain a global transition system D describing the joint behavior of the processes P_i. This global transition system is computed by constructing the reachable states of the product of the processes P_i. This product is the transition system $D = \langle AP, AC, S, \Delta, s_0, L \rangle$ where

- $AP = \bigcup_{1 \leq i \leq n} AP_i$.
- $AC = \bigcup_{1 \leq i \leq n} AC_i$.
- $S = \prod_{1 \leq i \leq n} S_i$. We denote the ith component of a state $s \in S$ by $s[i]$.
- $\langle s, a, s' \rangle \in \Delta$ if and only if
  - for all $1 \leq i \leq n$ such that $a \in AC_i$, we have $\langle s[i], a, s'[i] \rangle \in \Delta_i$, and
  - for all $1 \leq i \leq n$ such that $a \notin AC_i$, we have $s[i] = s'[i]$.
- $s_0 = \langle s_{0_1}, s_{0_2}, \ldots, s_{0_n} \rangle$.
- For every $s \in S$, we have $L(s) = \bigcup_i L_i(s[i])$.

We define the complexity of model checking for a concurrent program P with respect to the size of its components P_i and the length of the formula being checked. Accordingly, the program complexity of model checking for concurrent programs is defined in terms of the size of the components of the concurrent program.

Theorem 3.20 *The complexity of CTL and CTL* model checking for concurrent programs is PSPACE-complete.*

Proof: We have just proved that the model-checking problem for CTL and CTL* can be solved in space polynomial in the length of the formula but only poly-logarithmic in the size

(4.1) Search for a reachable state q' of the same Q_i that is provably false. If such a state q' is found, label q with 'F'.

(4.2) If no such state exists, search for a state $q' \in Q_i \cap B$ that is reachable from q and from itself. If such a state is found, label q is with 'F'.

(4.3) if none of the first two cases apply, label q with 'T'.

With every state q we can associate a finite integer, $rank(q)$, corresponding to the depth of the recursion required in order to label q. It is easy to prove, by induction on $rank(q)$, that q is labeled correctly. The procedure is recursive, but since each recursive call takes us to a lower Q_i, the depth of the recursion is bounded by m. Each invocation of the procedure can be executed nondeterministically using space $O(\log n)$. Therefore, by Savitch Theorem [Sav70], it can be executed deterministically in space $O(\log^2 n)$. Hence, the whole procedure can be executed using space $O(m \log^2 n)$. □

Theorems 3.12 and 3.14 provide us with sizes and depths of the HAAs associated with formulas of CTL and CTL*. Together with Proposition 3.2 and Theorem 3.17, we have the following.

Theorem 3.18

(1) The model-checking problem for CTL can be solved in space $O(m \log^2(mn))$, where m is the length of the formula and n is the size of the Kripke structure.

(2) The model-checking problem for CTL* can be solved in space $O(m(m + \log n)^2)$, where m is the length of the formula and n is the size of the Kripke structure.

It is interesting to note that a less space-efficient deterministic version of the algorithm given in the proof of Theorem 3.17 can be viewed as the automata-theoretic counterpart of the algorithm presented in [VL93].

Now, let us define the *program complexity* [VW86a] of model checking as the complexity of this problem in terms of the size of the input Kripke structure; i.e., assuming the formula fixed.

Theorem 3.19 *The program complexity of CTL and CTL* model checking is NLOGSPACE-complete.*

Proof: Fixing the formula, we get an HAA of a fixed depth. According to the algorithm presented in the proof of Theorem 3.17, the nonemptiness problem for such an HAA is in NLOGSPACE. Thus, so is the program complexity of CTL and CTL* model checking. Hardness in NLOGSPACE is immediate by a reduction from the graph accessibility problem, proved to be NLOGSPACE-complete in [Jon75]. □

graph of the automaton. We now show that by using a top-down exploration of this transition graph, we can get a space efficient 1-letter nonemptiness algorithm for HAAs.

Theorem 3.17 *The 1-letter nonemptiness problem for an HAA of size n and depth m, can be solved in space $O(m \log^2 n)$.*

Proof: The property of HAAs we use is that, from a state in Q_i, it is possible to search for another reachable state in the same Q_i nondeterministically using space $O(\log n)$. For transient Q_i, there are no such states. For universal and existential Q_i, the exact notion of reachability we use is the transitive closure of the following notion of *immediate reachability*. Consider a set Q_i and assume that we have a Boolean value for all states in sets lower than Q_i. Then, a state $q' \in Q$ is immediately reachable from a state q, if it appears in the transition from q when this transition has been simplified using the known Boolean values for states in lower Q_i's. Note that the simplified transition is always a disjunction for a state of an existential Q_i, and a conjunction for a state of a universal Q_i. A key step in our algorithm is the evaluation of Boolean expressions. It is known that this can be done using space that is logarithmic in the size of the expression [Lyn77]. Here, we evaluate expressions over Q, using a recursion of depth at most m to trace their Boolean value.

We call a state q' of Q_i *provably true* if, when the procedure is applied to the successors of q' that are not in Q_i, and the Boolean expression for the transition from q' is simplified, it is identically true. States that are *provably false* are defined analogously.

The following recursive procedure labels the states of the automaton with 'T' (accepts) or 'F' (does not accept).

(1) Start at an initial state.

(2) At a transient state q, evaluate the transition from q by recursively applying the procedure to the successor states of q. Label the state with the Boolean value that is obtained for the transition.

(3) At a state q of an existential Q_i, proceed as follows.

- **(3.1)** Search for a reachable state q' of the same Q_i that is provably true (note that this requires applying the procedure recursively to all states from lower Q_i's that are touched by the search). If such a state q' is found, label q with 'T'.
- **(3.2)** If no such state exists, search for a state $q' \in Q_i \cap G$ that is reachable from q and from itself. If such a state is found, label q with 'T'.
- **(3.3)** if none of the first two cases apply, label q with 'F'.

(4) At a state q of a universal Q_i, proceed as follows.

- The set Q_1 can not be a transient set. Consider the case where it is an existential set. Then, for every $q \in Q_1^1$, it must be that $\delta(q,a)$ is a disjunction of states in Q_1^1. The algorithm labels all the states in Q_1^1 with 'T' if and only if $Q_1^1 \cap G \neq \emptyset$. Assume first that $Q_1^1 \cap G \neq \emptyset$. Then, as Q_1^1 is a strongly connected component, there exists, for every $q \in Q_1^1$, a state $q' \in G$ such that q' is reachable from q and from itself. Hence, there exists a run of $\mathcal{A}^q$ that visits (in its single branch) the state q' infinitely often. This run is accepting an thus the language of $\mathcal{A}^q$ is not empty. Assume now that $Q_1^1 \cap G = \emptyset$. Then, for every $q \in Q_1^1$, no run of $\mathcal{A}^q$ can visit, even once, a state in G. Thus, no run of $\mathcal{A}^q$ is an accepting run. The proof is symmetric for the case Q_1 is a universal set.

- Assume that we have already labeled correctly all the states in all Q_k^l with $(k,l) < (i,j)$, and let $q \in Q_i^j$. If q is labeled before the phase (i,j) then, by the same consideration we used for WAAs, it is labeled correctly. Consider the case that q is labeled during the phase (i,j) and let Q_i be a universal set. Then, the simplification of $\delta(q,a)$ is a conjunction of states in Q_i^j and q it is labeled 'F' if and only if $Q_i^j \cap B \neq \emptyset$. Assume first that $Q_i^j \cap B \neq \emptyset$. Then, as Q_i^j is a strongly connected component, there exists, for every $q \in Q_i^j$, a state $q' \in B$ such that q' is reachable from q and from itself. Hence, in every run of $\mathcal{A}^q$ there is a copy that visits q' infinitely often. Thus, no run of $\mathcal{A}^q$ is an accepting run. Assume now that $Q_i^j \cap B = \emptyset$. Then, for every $q \in Q_i^j$, no copy of $\mathcal{A}^q$ can visit, even once, a state in B. So, every copy either stay in Q_i^j without visiting B, or reaches a state q' for which the language of $\mathcal{A}^{q'}$ is not empty. Thus, all the copies of $\mathcal{A}^q$ are accepting and the language of $\mathcal{A}^q$ is not empty. The proof is symmetric for the case Q_i is an existential set.

Since partitioning each graph into maximal strongly connected components can be done in linear running time [Tar72], the overall running time remains linear, as with WAAs. The data structure used for a linear-time implementation is similar to the one described for WAAs. Since the nonemptiness test for WAAs is described for simple WAAs, taking the nodes of the and/or graph to be the states of the WAA induced an and/or graph in a straightforward way. Here, however, $\mathcal{A}$ is not simple. Let $\mathcal{A}'$ be the simple alternating word automaton obtained from $\mathcal{A}$ by the simplification described in Theorem 3.1, and let $Q_1' \leq \ldots \leq Q_n'$ be the order on its sets (described there too). The automaton $\mathcal{A}'$ is no longer an HAA. Still, taking its states as the nodes of the and/or graph is proper: the flow of the labeling in the graph guarantees that when the algorithm reaches an existential (universal) set, all the nodes of this set which are still not labeled induce an or-graph (and-graph) as required. □

So, the 1-letter nonemptiness problem for HAAs can be solved in linear running time. This, together with Theorem 3.14, provides us with an exponential-time model-checking procedure for CTL*. Note that the algorithm used there is essentially a bottom-up labeling of the transition

Before we get to the space complexity of the 1-letter nonemptiness problem for HAAs, let us consider the time complexity of this problem.

Theorem 3.16 *The 1-letter nonemptiness problem for hesitant alternating automata is decidable in linear running time.*

Proof: Following Theorem 3.1, we present an algorithm with a linear running time for checking the nonemptiness of the language of an hesitant word alternating automaton $\mathcal{A} = \langle \{a\}, Q, \delta, Q^0, \langle G, B \rangle \rangle$. The algorithm is very similar to the one described in the proof of Theorem 3.9. The only change is that here we do not have accepting and rejecting sets. Rather, being trapped in an existential set, the automaton should be able to visit states from G infinitely often. Similarly, being trapped in a universal set, the automaton should be able to visit states from B only finitely often. As there, the algorithm proceeds up a total order of the Q_i's, labeling states with 'T' and 'F', guaranteeing that when it reaches a set Q_i, all the states in all sets Q_j's for which $Q_j < Q_i$, have already been labeled. For a transient set Q_i, the above implies that the states in it have already been labeled too. For existential and universal sets, the algorithm proceeds as follows.

- In an existential set Q_i, all the transitions from states in Q_i only contain disjunctively related elements of Q_i. So, when the algorithm reaches Q_i, all its simplified transitions are disjunctions and induce an or-graph that has states in Q_i as nodes. The algorithm partitions the graph into maximal strongly connected components Q_i^j. Since the partition is maximal, there exists a partial order over these component ($Q_i^j \leq Q_i^k$ if and only if there exists a transition from Q_i^k to Q_i^j) and there exists an extension of it into a total order $Q_i^1 \leq \ldots \leq Q_i^{m_i}$. The algorithm proceeds up this total order. When it reaches a component Q_i^j for which $Q_i^j \cap G \neq \emptyset$, then all the states in Q_i^j are labeled 'T'. Otherwise ($Q_i^j \cap G = \emptyset$), they are labeled 'F'. In both cases, the labeling is propagating. Indeed, a copy of $\mathcal{A}$ can stay in Q_i and visit states in G infinitely often if and only if it reaches a maximal strongly connected component Q_i^j, with $Q_i^j \cap G \neq \emptyset$.

- For a universal set, the transitions induce an and-graph. Accordingly, when the algorithm reaches a maximal strongly connected component Q_i^j for which $Q_i^j \cap B \neq \emptyset$, all the states in Q_i^j are labeled 'F' (and the labeling is propagating). Otherwise, they are labeled 'T'. Here, a copy of $\mathcal{A}$ can stay in Q_i and visits states in B only finitely often if and only if it reaches a component Q_i^j, with $Q_i^j \cap B = \emptyset$.

Consider the total order $Q_1^1 \leq Q_1^2 \leq \ldots Q_1^{m_1} \leq Q_2^1 \leq \ldots \leq Q_n^{m_n}$. We prove that for all $1 \leq i \leq n$ and $1 \leq j \leq m_i$, all the states in Q_i^j are labeled correctly. The proof proceeds by induction on (i, j) (with the ordering $(i_1, j_1) < (i_2, j_2)$ iff $i_1 < i_2$ or $i_1 = i_2$ and $j_1 < j_2$):

Starting with a sequential Büchi automaton $\mathcal{U}_\xi$ for $\xi = FG\varphi$, we construct $\mathcal{A}'_{\mathcal{D},E\xi}$.

$\mathcal{U}_\xi = \langle\{\{\varphi\},\emptyset\},\{q_0,q_1\},M,q_0,\{q_1\}\rangle$, with

- $M(q_0,\{\varphi\}) = \{q_0,q_1\}$
- $M(q_0,\emptyset) = \{q_0\}$
- $M(q_1,\{\varphi\}) = \{q_1\}$
- $M(q_1,\emptyset) = \emptyset$

Hence, $\mathcal{A}'_{\mathcal{D},E\xi} = \langle\{\{\varphi\},\emptyset\},\mathcal{D},\{q_0,q_1\},\delta',q_0,\langle\{q_1\},\emptyset\rangle\rangle$, with

q	$\delta'(q,\{\varphi\},k)$	$\delta'(q,\emptyset,k)$
q_0	$\bigvee_{c=0}^{k-1}((c,q_0)\vee(c,q_1))$	$\bigvee_{c=0}^{k-1}(c,q_0)$
q_1	$\bigvee_{c=0}^{k-1}(c,q_1)$	***false***

We are now ready to compose the automata into an automaton over the alphabet $\{\{p\},\emptyset\}$.

$\mathcal{A}_{\mathcal{D},E\xi} = \langle\{\{p\},\emptyset\},\mathcal{D},\{q_0,q_1,q_3,\tilde{q_3}\},\delta,q_0,\langle\{q_1\},\emptyset\rangle\rangle$, with

q	$\delta(q,\{p\},k)$	$\delta(q,\emptyset,k)$
q_0	$\bigvee_{c=0}^{k-1}(c,q_0)$	$\left(\bigvee_{c=0}^{k-1}((c,q_0)\vee(c,q_1))\wedge\bigvee_{c=0}^{k-1}(c,q_3)\right)\vee\left(\bigvee_{c=0}^{k-1}(c,q_0)\wedge\bigwedge_{c=0}^{k-1}(c,\tilde{q_3})\right)$
q_1	***false***	$\bigvee_{c=0}^{k-1}(c,q_1)\wedge\bigvee_{c=0}^{k-1}(c,q_3)$
q_3	***false***	***true***
$\tilde{q_3}$	***true***	***false***

Consider $\delta(q_0,\emptyset,k)$. The first conjunction corresponds to the case where $\mathcal{A}'_{\mathcal{D},E\xi}$ guesses that φ holds in the present. Then, $\mathcal{A}_{\mathcal{D},E\xi}$ proceeds with $\delta'(q_0,\{\varphi\},k)$ conjuncted with $\delta(q_2,\emptyset,k)$. The later guarantees that φ indeed holds in the present. The second conjunction corresponds to the case where φ does not hold in the present. Then, $\mathcal{A}_{\mathcal{D},E\xi}$ proceeds with $\delta'(q_0,\emptyset,k)$ conjuncted with $\delta(\tilde{q_2},\emptyset,k)$.

We obtain $\mathcal{A}_{\mathcal{D},\psi}$ by dualizing $\mathcal{A}_{\mathcal{D},E\xi}$. Hence, $\mathcal{A}_{\mathcal{D},\psi} = \langle\{\{p\},\emptyset\},\mathcal{D},\{\tilde{q_0},\tilde{q_1},\tilde{q_3},q_3\},\tilde{\delta},\tilde{q_0},\langle\emptyset,\{\tilde{q_1}\}\rangle\rangle$ with

q	$\tilde{\delta}(q,\{p\},k)$	$\tilde{\delta}(q,\emptyset,k)$
$\tilde{q_0}$	$\bigwedge_{c=0}^{k-1}(c,\tilde{q_0})$	$\left(\bigwedge_{c=0}^{k-1}((c,\tilde{q_0})\wedge(c,\tilde{q_1}))\vee\bigwedge_{c=0}^{k-1}(c,\tilde{q_3})\right)\wedge\left(\bigwedge_{c=0}^{k-1}(c,\tilde{q_0})\vee\bigvee_{c=0}^{k-1}(c,q_3)\right)$
$\tilde{q_1}$	***true***	$\bigwedge_{c=0}^{k-1}(c,\tilde{q_1})\vee\bigwedge_{c=0}^{k-1}(c,\tilde{q_3})$
$\tilde{q_3}$	***true***	***false***
q_3	***false***	***true***

Consider the state $\tilde{q_1}$. A copy of $\mathcal{A}_{\mathcal{D},\psi}$ that visits $\tilde{q_1}$ keeps creating new copies of $\mathcal{A}_{\mathcal{D},\psi}$, all visiting $\tilde{q_1}$. Spreading a copy that visits $\tilde{q_1}$ stops only when it reaches a node that satisfies p or AXp. Since $\tilde{q_1}\in B$, all the copies should eventually reach such a node. Hence, sending a copy that visits the state $\tilde{q_1}$ to a node x, $\mathcal{A}_{\mathcal{D},\psi}$ guarantees that all the paths in the subtree rooted x eventually reach a node satisfying $p\vee AXp$. Hence, in the state $\tilde{q_0}$, unless $\mathcal{A}_{\mathcal{D},\psi}$ gets convinced that $p\vee AXp$ holds in the present, it sends copies that visit $\tilde{q_1}$ to all the successors. In addition, it always sends copies visiting $\tilde{q_0}$ to all the successors.

proceeds according to an accepting run of $\mathcal{U}_\xi$ on a word that agrees with ρ on the formulas in $max(\varphi)$. Thus, $\rho \models \xi$ and $\langle T_K, V_K \rangle$ satisfies φ.

We now consider the size of $\mathcal{A}_\psi$. For every φ, we prove, by induction on the structure of φ, that the size of $\mathcal{A}_\varphi$ is exponential in $|\varphi|$.

- Clearly, for $\varphi = p$ or $\varphi = \neg p$ for some $p \in AP$, the size of $\mathcal{A}_\varphi$ is constant.
- For $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi = \varphi_1 \vee \varphi_2$, we have $|\mathcal{A}_\varphi| = O(|\mathcal{A}_{\varphi_1}| + |\mathcal{A}_{\varphi_2}|)$. By the induction hypothesis, $|\mathcal{A}_{\varphi_1}|$ is exponential in $|\varphi_1|$ and $|\mathcal{A}_{\varphi_2}|$ is exponential in $|\varphi_2|$. Thus, $|\mathcal{A}_\varphi|$ is surely exponential in $|\varphi|$.
- For $\varphi = E\xi$, we know, by [VW94], that the size of the word automaton $\mathcal{U}_\xi$ is exponential in $|\xi|$. Therefore, $\mathcal{A}'_\varphi$ is exponential in $|\varphi|$. Also, $|\Sigma'|$ is exponential in $|max(\varphi)|$ and, by the induction hypothesis, for all $\varphi_i \in max(\varphi)$, the size of $\mathcal{A}_{\varphi_i}$ is exponential in $|\varphi_i|$. Therefore, $\mathcal{A}_\varphi$ is also exponential in $|\varphi|$.
- For $\varphi = A\xi$, we know, by the above, that $|\mathcal{A}_{E\neg\xi}|$ is exponential in $|\varphi|$. Since complementing an HAA does not change its size, the result for φ follows.

Finally, since each subformula of ψ induces exactly one set, the depth of $\mathcal{A}_\psi$ is linear in $|\psi|$. □

We note that the same construction holds also for ECTL* [VW84]. In ECTL*, formulas are constructed from Boolean connectives and automata connectives. As the construction above handles a formula $E\xi$ by translating the path formula ξ into an automaton, allowing automaton operators in the path formulas causes no difficulty. We handle these automaton operators as we handle the word automaton $\mathcal{U}_\xi$ constructed for a CTL* path formula ξ.

Example 3.15 Consider the CTL* formula $\psi = AGF(p \vee AXp)$. We describe the construction of $\mathcal{A}_{\mathcal{D},\psi}$, for all $\mathcal{D} \subset \mathbb{N}$, step by step. Since ψ is of the form $A\xi$, we need to construct and dualize the HAA of the formula $EFG((\neg p) \wedge EX\neg p)$. We start with the HAAs $\mathcal{A}_{\mathcal{D},\varphi}$ and $\tilde{\mathcal{A}}_{\mathcal{D},\varphi}$ for $\varphi = (\neg p) \wedge EX\neg p$.

$\mathcal{A}_{\mathcal{D},\varphi} = \langle \{\{p\}, \emptyset\}, \mathcal{D}, \{q_2, q_3\}, \delta, q_2, \langle \emptyset, \emptyset \rangle \rangle$, with

q	$\delta(q, \{p\}, k)$	$\delta(q, \emptyset, k)$
q_2	***false***	$\bigvee_{c=0}^{k-1}(c, q_3)$
q_3	***false***	***true***

$\tilde{\mathcal{A}}_{\mathcal{D},\varphi} = \langle \{\{p\}, \emptyset\}, \mathcal{D}, \{\tilde{q_2}, \tilde{q_3}\}, \tilde{\delta}, \tilde{q_2}, \langle \emptyset, \emptyset \rangle \rangle$, with

q	$\tilde{\delta}(q, \{p\}, k)$	$\tilde{\delta}(q, \emptyset, k)$
$\tilde{q_2}$	***true***	$\bigwedge_{c=0}^{k-1}(c, \tilde{q_3})$
$\tilde{q_3}$	***true***	***false***

$\{q_0\}$ constitutes a transient set, with the ordering $\{q_0\} > Q_i$ for all the sets Q_i in Q^1 and Q^2.

The construction for $\varphi = \varphi_1 \vee \varphi_2$ is similar, with $\delta(q_0, \sigma) = \delta(q_0^1, \sigma) \vee \delta(q_0^2, \sigma)$.

- If $\varphi = E\xi$, where ξ is a CTL* path formula, we first build an HAA, $\mathcal{A}'_\varphi$, over the alphabet $\Sigma' = 2^{max(\varphi)}$. That is, $\mathcal{A}'_\varphi$ regards the formulas maximal in φ as atomic propositions. Let $\mathcal{U}_\xi = \langle \Sigma', Q, M, q_0, F \rangle$ be a nondeterministic Büchi automaton on infinite words such that $\mathcal{U}_\xi$ accepts exactly all the word models of ξ [VW94]. Then, $\mathcal{A}'_\varphi = \langle \Sigma', Q, \delta', q_0, \langle F, \emptyset \rangle \rangle$ extends $\mathcal{U}_\xi$ by tracing it in a single branch. Precisely, for all $q \in Q$ and $\sigma' \in \Sigma'$, we have

$$\delta'(q, \sigma') = \bigvee_{q_i \in M(q, \sigma')} (0, q_i) \vee (1, q_i).$$

If $M(q, \sigma') = \emptyset$, then $\delta'(q, \sigma') = \textbf{\textit{false}}$. Note that the disjunction above is the only place where $\mathcal{D}$ plays a role. Note also that Q constitutes a single and existential set. The HAA $\mathcal{A}'_\varphi$ accepts exactly all the Σ'-labeled trees that satisfy φ.

We now adjust $\mathcal{A}'_\varphi$ to the alphabet Σ. The resulted automaton is $\mathcal{A}_\varphi$. Intuitively, $\mathcal{A}_\varphi$ sends additional copies to HAAs associated with formulas in $max(\varphi)$. These copies guarantee that whenever $\mathcal{A}'_\varphi$ assumes that a formula in $max(\varphi)$ holds, then it indeed holds, and that whenever $\mathcal{A}'_\varphi$ assumes that a formula does not hold, then the negation of the formula holds. Formally, $\mathcal{A}_\varphi = \langle \Sigma, Q \cup \bigcup_i (Q^i \cup \tilde{Q}^i), \delta, q_0, \langle F \cup \bigcup_i (G^i \cup \tilde{G}^i), \bigcup_i (B^i \cup \tilde{B}^i) \rangle \rangle$, where δ is defined as follows. For states in $\bigcup_i (Q^i \cup \tilde{Q}^i)$, it agrees with the corresponding δ^i and $\tilde{\delta}^i$. For $q \in Q$ and for all $\sigma \in \Sigma$, we have

$$\delta(q, \sigma) = \bigvee_{\sigma' \in \Sigma'} (\delta'(q, \sigma') \wedge (\bigwedge_{\varphi_i \in \sigma'} \delta^i(q_0^i, \sigma)) \wedge (\bigwedge_{\varphi_i \notin \sigma'} \tilde{\delta}^i(\tilde{q}_0^i, \sigma))).$$

Each conjunction in δ corresponds to a label $\sigma' \in \Sigma'$. Some copies of $\mathcal{A}_\varphi$ (these originated from $\delta'(q, \sigma')$) proceed as $\mathcal{A}'_\varphi$ when it reads σ'. Other copies guarantee that σ' indeed holds in the current node. The set Q constitutes an existential set, with the ordering $Q > Q'$ for all the sets Q' in $\bigcup_i (\{Q^i\} \cup \{\tilde{Q}^i\})$.

- If $\varphi = A\xi$, we construct and dualize the HAA of $E\neg\xi$.

We prove the correctness of the construction by induction on the structure of φ. The proof is immediate for the case φ is of the form p, $\neg p$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, or $A\xi$. We consider here the case where $\varphi = E\xi$. If a tree $\langle T_K, V_K \rangle$ satisfies φ, then there exists a path ρ in it such that $\rho \models \xi$. Thus, there exists an accepting run r of $\mathcal{U}_\xi$ on a word that agrees with ρ on the formulas in $max(\varphi)$. Clearly, a run of $\mathcal{A}_\varphi$ that proceeds on ρ according to r accepts $\langle T_K, V_K \rangle$. Now, if a run r of $\mathcal{A}_\varphi$ accepts a tree $\langle T_K, V_K \rangle$, then there must be a path ρ in this tree such that $\mathcal{A}_\varphi$

Translating an acceptance condition $\langle G, B\rangle$ of an HAA into the Muller acceptance condition, one gets in α sets S for which $S \subseteq Q_i$ for some Q_i and either Q_i is an existential set, in which case $S \cap G \neq \emptyset$, or Q_i is a universal set, in which case $S \cap B = \emptyset$. Hence, dualizing this α, one gets in $\tilde{\alpha}$ sets S for which there exists no Q_i such that $S \subseteq Q_i$, or, if such Q_i exists, then either it is existential and $S \cap G = \emptyset$, or it is universal and $S \cap B \neq \emptyset$. Since each path must get trapped in some Q_i, we need in $\tilde{\alpha}$ only sets S for which $S \subseteq Q_i$ for some Q_i. So, the only relevant sets in $\tilde{\alpha}$ are S for which either $S \subseteq Q_i$ for an existential Q_i, in which case $S \cap G = \emptyset$, or $S \subseteq Q_i$ for a universal Q_i, in which case $S \cap B \neq \emptyset$. Dualizing δ, existential sets become universal and vice versa. Hence, going back to an HAA we get the acceptance condition $\langle B, G\rangle$. □

For an HAA $\mathcal{A}$, we say that $\tilde{\mathcal{A}}$ is the *dual* HAA of $\mathcal{A}$.

Theorem 3.14 *Given a CTL* formula ψ and a set $\mathcal{D} \subset \mathbb{N}$, we can construct an HAA $\mathcal{A}_{\mathcal{D},\psi}$ of size $O(|\mathcal{D}| \cdot 2^{|\psi|})$ and of depth $O(|\psi|)$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ.*

Proof: Before defining $\mathcal{A}_{\mathcal{D},\psi}$ we need the following definitions and notations. For two CTL* formulas θ and φ, we say that θ is *maximal* in φ, if and only if θ is a strict subformula of φ and there exists no formula "between them". Namely, there exists no strict subformula ξ of φ such that θ is a strict subformula of ξ. We denote by $max(\varphi)$ the set of all formulas maximal in φ. For example, $max(A(Xp)U(EXq)) = \{p, EXq\}$.

We construct $\mathcal{A}_{\mathcal{D},\psi}$ by induction on the structure of ψ (assuming it is in a positive normal form). For technical convenience we describe the definition for binary trees and we assume that each HAA has a single initial state. The extension to any $\mathcal{D} \subset \mathbb{N}$ and to any set of initial states is straightforward. With each formula $\varphi \in cl(\psi)$, we associate an HAA $\mathcal{A}_\varphi$ composed from HAAs associated with formulas maximal in φ. We assume that the state sets of composed HAAs are disjoint (otherwise, we rename states) and that for all the HAAs, we have $\Sigma = 2^{AP}$ (that is, an HAA associated with a subformula that does not involve all AP is extended). For φ with $max(\varphi) = \{\varphi_1, \ldots, \varphi_n\}$ and for all $1 \leq i \leq n$, let $\mathcal{A}_{\varphi_i} = \langle \Sigma, Q^i, \delta^i, q_0^i, \langle G^i, B^i\rangle\rangle$ be the HAA associated with φ_i and let $\tilde{\mathcal{A}_{\varphi_i}} = \langle \Sigma, \tilde{Q^i}, \tilde{\delta^i}, \tilde{q_0^i}, \langle \tilde{G^i}, \tilde{B^i}\rangle\rangle$ be its dual HAA. We define $\mathcal{A}_\varphi$ as follows.

- If $\varphi = p$ or $\varphi = \neg p$ for some $p \in AP$, then $\mathcal{A}_\varphi$ is a one-state HAA.
- If $\varphi = \varphi_1 \wedge \varphi_2$, then $\mathcal{A}_\varphi = \langle \Sigma, Q^1 \cup Q^2 \cup \{q_0\}, \delta, q_0, \langle G^1 \cup G^2, B^1 \cup B^2\rangle\rangle$, where q_0 is a new state and δ is defined as follows. For states in Q^1 and Q_2, the transition function δ agrees with δ^1 and δ^2. For the state q_0 and for all $\sigma \in \Sigma$, we have $\delta(q_0, \sigma) = \delta(q_0^1, \sigma) \wedge \delta(q_0^2, \sigma)$. Thus, in the state q_0, $\mathcal{A}_\varphi$ sends all the copies sent by both $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$. The singleton

the translation of CTL formulas to HAAs.

Theorem 3.12 *Given a CTL formula ψ and a set $\mathcal{D} \subset \mathbb{N}$, we can construct an HAA $\mathcal{A}_{\mathcal{D},\psi}$ of size $O(|\mathcal{D}| \cdot |\psi|)$ and of depth $O(|\psi|)$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees models satisfying ψ.*

Proof: As observed above, each set in the WAAs that correspond to CTL formulas is either transient, existential, or universal. Thus, we only have to define a suitable acceptance condition. In the WAAs, we allow a path to get trapped in a set that corresponds to $\tilde{U}$-formulas. Accordingly, in HAAs, we allow a path to get trapped in an existential set only if it corresponds to a $\tilde{U}$-formula and we allow a path to get trapped in a universal set only if it does not correspond to an U-formula. This is done with the acceptance condition $\langle G, B\rangle$ where G is the set of all $E\tilde{U}$-formulas in $cl(\psi)$ and B is the set of all AU-formulas in $cl(\psi)$. Since each set in the HAA corresponds to a single formula in $cl(\psi)$, the depth of the HAA is at most $|\psi|$. □

We now present a translation of CTL* formulas to HAAs. Weak alternating automata define exactly the set of weakly definable languages [MSS86]. The logic CTL* can define languages that are not weakly definable. For example, the set of trees that satisfy the CTL* formula $AFGp$ is not weakly definable [Rab70]. Therefore, a stronger acceptance condition is required for automata corresponding to formulas of CTL*. As we shall see later, the stronger acceptance condition does not harm the complexity of the 1-letter nonemptiness problem. We first show that complementation is easy for HAAs. For two HAA $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same alphabet Σ, we say that $\mathcal{A}_1$ *complements* $\mathcal{A}_2$ iff $\mathcal{L}(\mathcal{A}_1)$ includes exactly all the Σ-labeled trees which are not in $\mathcal{L}(\mathcal{A}_2)$.

Given a transition function δ, let $\tilde{\delta}$ denote the dual function of δ. That is, for every q, σ, and k, with $\delta(q,\sigma,k) = \theta$, let $\tilde{\delta}(q,\sigma,k) = \tilde{\theta}$, where $\tilde{\theta}$ is obtained from θ by switching $\vee$ and $\wedge$ and by switching ***true*** and ***false***. If, for example, $\theta = p \vee (\textbf{\textit{true}} \wedge q)$ then $\tilde{\theta} = p \wedge (\textbf{\textit{false}} \vee q)$,

Lemma 3.13 *Given an HAA $\mathcal{A} = \langle \Sigma, Q, \delta, Q^0, \langle G, B\rangle\rangle$, the alternating automaton $\tilde{\mathcal{A}} = \langle \Sigma, Q, \tilde{\delta}, Q^0, \langle B, G\rangle\rangle$ is an HAA and $\tilde{\mathcal{A}}$ complements $\mathcal{A}$.*

Proof: It is easy to see that $\tilde{\mathcal{A}}$ is an HAA. Indeed, the partition of Q into sets and the partial order over them hold also with respect to $\tilde{\mathcal{A}}$. In particular, a set that is existential in $\mathcal{A}$ is universal in $\tilde{\mathcal{A}}$ and vise versa. In [MS87], Muller and Schupp prove that dualizing an alternating automaton complements it. There, the acceptance condition used is $\alpha \subseteq 2^Q$ (the Muller acceptance condition). In order to satisfy a Muller acceptance condition α in a run r, a path ρ should have $Inf(r|\rho) \in \alpha$. Dualizing this acceptance condition is taking $\tilde{\alpha} = 2^Q \setminus \alpha$ instead α. We show that translating $\mathcal{A}$ into a Muller automaton, dualizing this automaton, and translating it back into an HAA, results in $\tilde{\mathcal{A}}$.

is at most one element of Q_i in each conjunct). These are the Q_i's corresponding to the $A\varphi_1 U \varphi_2$ and $A\varphi_1 \tilde{U} \varphi_2$ elements of $cl(\psi)$.

This means that it is only when moving from one Q_i to the next, we can move from a state that is conjunctively related to states in its set to a state that is disjunctively related to states in its set, or vice-versa. In other words, when a copy of the automaton visits a state in some set Q_i which is associated with an EU-formula of an $E\tilde{U}$-formula, then as long as it stays in this set, it proceeds in an "existential mode"; namely, it imposes only existential requirements on its successors in Q_i. Similarly, when a copy of the automaton visits a state in some set Q_i which is associated with an AU-formula or an $A\tilde{U}$-formula, then as long as it stays in this set, it proceeds in a "universal mode". Thus, whenever a copy alternates modes, it must be that it moves from one Q_i to a lower one.

The above observation is captured in the restricted structure of hesitant alternating automata (HAAs), and is the key to our space-efficient model-checking procedure for CTL and CTL*. An HAA is an alternating automaton $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, \delta, Q^0, \alpha \rangle$, where $\alpha = \langle G, B \rangle$ with $G \subseteq Q$ and $B \subseteq Q$. That is, the acceptance condition of HAAs consists of a pair of sets of states. As in WAAs, there exists a partition of Q into disjoint sets and a partial order $\leq$ such that transitions from a state in Q_i lead to states in either the same Q_i or a lower one. In addition, each set Q_i is classified as either *transient*, *existential*, or *universal*, such that for each set Q_i and for all $q \in Q_i$, $\sigma \in \Sigma$, and $k \in \mathcal{D}$, the following hold:

1. If Q_i is a transient set, then $\delta(q, \sigma, k)$ contains no elements of Q_i.
2. If Q_i is an existential set, then $\delta(q, \sigma, k)$ only contains disjunctively related elements of Q_i.
3. If Q_i is a universal set, then $\delta(q, \sigma, k)$ only contains conjunctively related elements of Q_i.

It follows that every infinite path ρ of a run r gets trapped within some either an existential or a universal set Q_i. The path then satisfies an acceptance condition $\langle G, B \rangle$ if and only if either Q_i is an existential set and $Inf(r|\rho) \cap G \neq \emptyset$, or Q_i is a universal set and $Inf(r|\rho) \cap B = \emptyset$. Note that the acceptance condition of HAAs combines the Rabin and the Streett acceptance conditions: existential sets refer to a Rabin condition $\{\langle G, \emptyset \rangle\}$ and universal sets refer to a Street condition $\{\langle B, \emptyset \rangle\}$. Note also that while the transition function of HAAs is more restricted than the one of WAAs, their acceptance condition is more expressive. We will need the stronger acceptance condition to handle CTL* formulas. The number of sets Q_i of an HAA is defined as the *depth* of the HAA.

3.4.2 The Space Complexity of CTL and CTL* Model Checking

In Theorem 3.3 we presented a translation of CTL formulas to WAAs. We have already shown that the resulted WAAs have the restricted structure of HAAs. Theorem 3.12 below completes

the emptiness of a Büchi automaton on words which is NLOGSPACE-complete. Thus, if the Büchi automaton whose emptiness has to be checked is obtained as the product of the components of a concurrent program (as is usually the case), the space required is polynomial in the size of these components rather than of the order of the exponentially larger Büchi automaton. Pragmatically, this is very significant and is, to some extent, exploited in the "on the fly" approaches to model checking and in related memory saving techniques [CVWY92, MP94].

Is the same true of branching-time model-checking? The answer to this question was long thought to be negative. Indeed, the bottom-up nature of the known model-checking algorithms seemed to imply that storing the whole structure was required. Using our automata-theoretic approach to branching-time model-checking, we are able to show that this is not necessarily so. In this section we introduce a new type of alternating automata, called *hesitant alternating automata* (HAAs), for which the 1-letter nonemptiness problem can be solved in a very efficient space. We show that formulas of CTL and CTL* can be translated to HAAs and that the model-checking problem for these logics can be solved in space polynomial in $m \log n$, where m is the length of the formula and n is the size of the Kripke structure. Hence, the model-checking problem for concurrent programs for these logics is in PSPACE. We claim that the ability to translate formulas to HAAs is of a great importance when space complexity of model checking is considered. For example, formulas of the alternation-free μ-calculus can not be translated to HAAs and the model-checking problem for concurrent programs for this logic is EXPTIME-complete.

3.4.1 Hesitant Alternating Automata

Consider the product automaton $\mathcal{A}_{K,\psi} = K \times \mathcal{A}_{\mathcal{D},\psi}$ for a Kripke structure K and a CTL formula ψ. The states of $\mathcal{A}_{K,\psi}$ are elements of $W \times cl(\psi)$ and they are partitioned into sets Q_i according to their second component (two states are in the same Q_i if and only if their second components are identical). Thus, the number of Q_i's is bounded by the size of $cl(\psi)$ and is independent of the size of the Kripke structure. If we examine the Q_i's closely, we notice that they all fall into one of the following three categories:

1. Sets from which all transitions lead exclusively to states in lower Q_i's. These are the Q_i's corresponding to all elements of $cl(\psi)$ except U-formulas and $\tilde{U}$-formulas.

2. Sets Q_i such that, for all $q \in Q_i$, the transition $\delta(q, a, k)$ only contains *disjunctively related* elements of Q_i (i.e., if the transition is rewritten in disjunctive normal form, there is at most one element of Q_i in each disjunct). These are the Q_i's corresponding to the $E\varphi_1 U \varphi_2$ and $E\varphi_1 \tilde{U} \varphi_2$ elements of $cl(\psi)$.

3. Sets Q_i such that, for all $q \in Q_i$, the transition $\delta(q, a, k)$ only contains *conjunctively related* elements of Q_i (i.e., if the transition is rewritten in conjunctive normal form, there

3.3.2 Model Checking for the μ-Calculus

The intimate connection between the μ-calculus and alternating automata has been noted in [EJ91, Eme94]. We show here that our automata-theoretic approach provides a clean proof that model checking for the μ-calculus is in NP$\cap$co-NP [EJS93]. The key steps in the proof are showing that μ-calculus formulas can be efficiently translated to alternating Rabin automata, and that the 1-letter nonemptiness problem for alternating Rabin automata is in NP.

Theorem 3.10 *Given a μ-calculus formula ψ and a set $\mathcal{D} \subset I\!N$, we can construct, in linear running-time, an alternating Rabin automaton $\mathcal{A}_{\mathcal{D},\psi}$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ.*

Proof: Emerson and Jutla showed how to translate, in linear running-time, μ-calculus formulas to alternating *Streett* automata [EJ91]. By constructing an alternating Streett automaton for $\neg\psi$ and then complementing it (it is easy to complement alternating automata [MS87]), we obtain an alternating Rabin automaton for ψ. □

Theorem 3.11 *The 1-letter nonemptiness problem for alternating Rabin automata is decidable in nondeterministic polynomial running time.*

Proof: According to Theorem 3.1, the 1-letter nonemptiness problem for alternating Rabin automata is of the same complexity as the nonemptiness problem for nondeterministic Rabin tree automata. By [Eme85, VS85], the later is in NP. □

Combining Theorems 3.10 and 3.11, Proposition 3.2, and the observation in [EJS93] that checking for satisfaction of a formula ψ and a formula $\neg\psi$ has the same complexity, we get that the model-checking problem for the μ-calculus is in NP$\cap$co-NP.

3.4 The Space Complexity of Model Checking

Lichtenstein and Pnueli argued that when analyzing the complexity of model checking, a distinction should be made between complexity in the size of the input structure and complexity in the size of the input formula; it is the complexity in size of the structure that is typically the computational bottleneck [LP85]. The Kripke structures to which model-checking is applied are often obtained by constructing the reachability graph of *concurrent programs*, and can thus be very large. So, even linear complexity, in terms of the input structure, can be excessive, especially as far as space is concerned. The question is then whether it is possible to perform model-checking without ever holding the whole structure to be checked in memory at any one time. For linear temporal formulas, the answer is positive [VW86a]. Indeed, this problem reduces to checking

Using an and/or graph, as suggested in [Bee80], the algorithm can be implemented in linear running time. The graph, G, induced by the transition function, maintains the labeling and the propagation of labeling performed during the algorithm execution. In more details, each node of G corresponds to a state $q \in Q$. For q with $\delta(q,a) = q_1 * q_2$, the node q is a $*$-node with two successors, q_1 and q_2. For $q = \{\textbf{\textit{true}}, \textbf{\textit{false}}\}$, the node q is a sink-node. Since $\mathcal{A}$ is simple, these are the only possible forms of transitions and hence the only possible nodes. Each $*$-node q is denoted by a triple $\langle *, last, ptrs \rangle$, where $last$ is a Boolean flag and $ptrs$ is a set of pointers that point to nodes that have q as a successor. The Boolean flag $last$ is true iff one of the two successors of q has already been labeled with 'T' or 'F', its labeling has been propagated to q, but did not suffice to label q as well. In the beginning, $last = \textbf{\textit{false}}$ for all the nodes. The sink-node q is denoted by a set $ptrs$ of pointers that point to nodes that have q as a successor. It is easy to see that the size of G is linear in the size of δ.

In addition, the algorithm maintains an integer i that contains the current phase and two stacks S_T and S_F. The stacks contain nodes that were labeled with 'T' and 'F', yet still have not propagated their labeling further. In the beginning, $i = 1$, the stack S_T contains the sink-node $\textbf{\textit{true}}$ (if exists), and the stack S_F contains the sink-node $\textbf{\textit{false}}$ (if exists).

Using G, the algorithm proceeds as follows. Whenever a node in the graph is labeled with 'T' ('F'), the node is pushed into S_T (S_F). As long as S_T or S_F are not empty, some node q is popped from, say, S_T and for every node q' that has q as a successor (as detected by $ptrs$), the algorithm proceeds as follows (handling of $q \in S_F$ is dual):

- If q' is a $\vee$-node, then label q' with 'T'.
- If q' is a $\wedge$-node with $last = \textbf{\textit{true}}$, then label q' with 'T'.
- If q' is a $\wedge$-node with $last = \textbf{\textit{false}}$, then change $last$ to $\textbf{\textit{true}}$.

When both S_T and S_F are empty, nodes that correspond to the states of the current Q_i are labeled according to the classification of Q_i and i is increased. Since each node of G is pushed into a stack only once, and since handling of a node that is popped from a stack involves a constant number of operations to each of the nodes that have it as a successor, the entire complexity is linear in the size of G; hence linear in the size of δ. □

Theorems 3.3, 3.6, and 3.9, together with Proposition 3.2, yield model-checking algorithms for CTL and for the alternation-free μ-calculus, with linear (in the size of the input structure and in the size of the input formula) running time. The bottom-up labeling of the algorithm used in the proof of Theorem 3.9 is clearly reminiscent of the bottom-up labeling that takes place in the standard algorithms for CTL and alternation-free μ-calculus model checking [CES86, Cle93]. Thus, the automata-theoretic approach seems to capture the combinatorial essence of branching-time model checking.

i.e., a conjunction with a conjunct 'F' is simplified to 'F' and a disjunction with a disjunct 'T' is simplified to 'T'. Consequently, a transition function $\delta(q', a)$ for some q' (not necessarily from Q_i) can be simplified to ***true*** or ***false***. The state q' is then labeled, and simplification propagates further.

Since the algorithm proceeds up the total order, when it reaches a state $q \in Q_i$ that is still not labeled, it is guaranteed that all the states in all Q_j for which $Q_j < Q_i$, have already been labeled. Hence, all the states that occur in $\delta(q, a)$ have the same status as q. That is, they belong to Q_i and are still not labeled. The algorithm then labels q and all the states in $\delta(q, a)$ according to the classification of Q_i. They are labeled 'T' if Q_i is accepting and are labeled 'F' otherwise.

Correct operation of the algorithm can be understood as follows. It is guaranteed that once the automaton visits a state that belongs to Q_1, it visits only states from Q_1 thereafter. Similarly, when the automaton visits a state q whose labeling can not be decided according to labeling of states in lower sets, this state leads to a cycle or belongs to a cycle of states of the same status. Hence the labeling of states according to the classification of the set they belong to.

Formally, we prove that for all $1 \leq i \leq n$, all the states in Q_i are labeled correctly. The proof proceeds by induction on i. The case $i = 1$ is immediate. Assume that we have already labeled correctly all the states in all Q_j with $j < i$ and let $q \in Q_i$. We consider the case where Q_i is an accepting set. The proof is symmetric for the case Q_i is a rejecting set. We distinguish between three possibilities of labeling q:

1. The state q is labeled 'T' before the phase i. Then, the value of $\delta(q, a)$, simplified according to the labeling already done, is ***true***. Therefore, there exists a run of $\mathcal{A}^q$ in which every copy created in the first step reaches a state q' for which, by the induction hypothesis, the language of $\mathcal{A}^{q'}$ is not empty. Hence, the language of $\mathcal{A}^q$ is also not empty.

2. The state q is labeled 'F' before the phase i. The correctness proof is symmetric to the one of the previous case.

3. The state q is labeled 'T' during the phase i. Then, it must be the case that the simplification of $\delta(q, a)$ contains states of Q_i. Moreover, it contains only states of Q_i and they all have not been labeled before the phase i. Thus, there exists a run of $\mathcal{A}^q$ in which every copy created in the first step either reaches a state q' for which the language of $\mathcal{A}^{q'}$ is not empty, or stays forever in Q_i. Hence, the language of $\mathcal{A}^q$ is not empty.

Note that these are indeed the only possibilities of labeling q: a state in an accepting set Q_i can not be labeled after the phase i and it can not be labeled with 'F' during the phase i.

We now turn to study the time complexity of the 1-letter nonemptiness problem for WAAs. We first consider the general nonemptiness problem for them.

Theorem 3.8 *The nonemptiness problem for weak alternating automata is EXPTIME-complete.*

Proof: Membership in EXPTIME is proved in [MSS86]. Hardness in EXPTIME follows from reduction of satisfiability of CTL, proved to be EXPTIME-hard in [FL79]. □

Thus, the general nonemptiness problem for WAAs can not be solved efficiently. Nevertheless, as we prove below, the 1-letter nonemptiness for WAAs can be solved in linear time. Note that, as follows from Theorem 3.1, the best upper-bound known for 1-letter nonemptiness of Buchi alternating automata is quadratic [VW86a]. Thus, the weakness of the automaton is essential.

Theorem 3.9 *The 1-letter nonemptiness problem for weak alternating automata is decidable in linear running time.*

Proof: Following Theorem 3.1, we prove that the 1-letter nonemptiness problem for simple weak alternating word automata is decidable in linear running time. We present an algorithm with linear running time for checking the nonemptiness of the language of a simple weak alternating word automaton $\mathcal{A} = \langle \{a\}, Q, \delta, Q^0, \alpha \rangle$.

The algorithm labels the states of $\mathcal{A}$ with either 'T', standing for ***true***, or 'F', standing for ***false***. Typically, states $q \in Q$ for which the language of $\mathcal{A}^q$ (i.e., the language of $\mathcal{A}$ with q as the initial state) is nonempty are labeled with 'T' and states q for which the language of $\mathcal{A}^q$ is empty are labeled with 'F'. The language of $\mathcal{A}$ is thus nonempty if and only if the initial state q_0 is labeled with 'T'.

As $\mathcal{A}$ is weak, there exists a partition of Q into disjoint sets Q_i such that there exists a partial order $\leq$ on the collection of the Q_i's and such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, a)$, we have that $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. In addition, each set Q_i is classified as accepting, if $Q_i \subseteq \alpha$, or rejecting, if $Q_i \cap \alpha = \emptyset$. The algorithm works in phases and proceeds up the partial order. We regard ***true*** and ***false*** as states with a self loop. The state ***true*** constitutes an accepting set and the state ***false*** constitutes a rejecting set, both minimal in the partial order. Let $Q_1 \leq \ldots \leq Q_n$ be an extension of the partial order to a total order. In each phase i, the algorithm handles states from the minimal set Q_i that still has not been labeled.

States that belong to the set Q_1 are labeled according to the classification of Q_1. Thus, they are labeled with 'T' if Q_1 is an accepting set, and with 'F' if it is rejecting. Once a state $q \in Q_i$ is labeled with 'T' or 'F', transition functions in which q occurs are simplified accordingly;

Theorem 3.6 *Given an alternation-free guarded μ-calculus formula ψ and a set $\mathcal{D} \subset \mathbb{N}$, we can construct in linear running time, a WAA $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, cl(\psi), \delta, \psi, \alpha \rangle$, such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ.*

Proof: For atomic propositions constants and for formulas of the forms $\varphi_1 \wedge \varphi_2, \varphi_1 \wedge \varphi_2, AX\varphi_2$, or $EX\varphi_2$, the transition function δ is equal to the one described for CTL. For μ and ν formulas, and for all $\sigma \in \Sigma$ and $k \in \mathcal{D}$, we define:

- $\delta(\mu y.f(y), \sigma, k) = \delta(f(\mu y.f(y)), \sigma, k)$.
- $\delta(\nu y.f(y), \sigma, k) = \delta(f(\nu y.f(y)), \sigma, k)$.

Note that since ψ is guarded, it is guaranteed that we have no circularity in the definition of δ. In order to define α, we define an equivalence relation $\mathcal{R}$ over $cl(\varphi)$ where

$$\varphi_1 \mathcal{R} \varphi_2 \text{ iff } \varphi_1 \in cl(\varphi_2) \text{ and } \varphi_2 \in cl(\varphi_1).$$

Since φ is alternation free, each equivalence class of $\mathcal{R}$ can not contain both a ν-formula and a μ-formula. A state $\varphi \in cl(\psi)$ belongs to α if and only if it belongs to an equivalence class that contains a ν-formula.

To show that $\mathcal{A}_{\mathcal{D},\psi}$ is a WAA, we use $\mathcal{R}$ as the required partition of Q into disjoint sets. Each equivalence class of $\mathcal{R}$ constitutes a set Q_i. We denote each set Q_i by $[\varphi]$, for some $\varphi \in Q_i$. The partial order is defined by $[\varphi_1] \leq [\varphi_2]$ iff $\varphi_1 \in cl(\varphi_2)$. As in CTL, since each transition of the automaton from a state φ leads to states associated with formulas in $cl(\varphi)$, the weakness conditions hold. In particular, each set is either contained in α or disjoint from α.

The correctness proof of the construction is similar to the one for CTL. Here, the definition of α guarantees that an accepting run can not get trapped in a set with a μ-formula, and, on the other hand, it is allowed to stay forever in a set with a ν-formula. □

Example 3.7 Consider the formula $\psi = \mu y.(p \vee EXAXy)$. For every $\mathcal{D} \subset \mathbb{N}$, the WAA associated with ψ and $\mathcal{D}$ is $\mathcal{A}_{\mathcal{D},\psi} = \langle \{\{p\}, \emptyset\}, \mathcal{D}, \{\psi, AX\psi\}, \delta, \psi, \emptyset \rangle$, where δ is described below (we restrict $\mathcal{A}_{\mathcal{D},\psi}$ to its reachable states).

By the definition of δ, we have that $\delta(\mu y.(p \vee EXAXy), \sigma, k) = \delta(p \vee EXAX\mu y.(p \vee EXAXy), \sigma, k)$. Hence we have:

q	$\delta(q, \{p\}, k)$	$\delta(q, \emptyset, k)$
ψ	***true***	$\bigvee_{c=0}^{k-1}(c, AX\psi)$
$AX\psi$	$\bigwedge_{c=0}^{k-1}(c, \psi)$	$\bigwedge_{c=0}^{k-1}(c, \psi)$

In the state ψ, if p does not hold in the present, $AX\psi$ should be satisfied in some successor. Since the state set of $\mathcal{A}_{\mathcal{D},\psi}$ constitutes a single, and rejecting, set, p should eventually hold.

(note that for every run, ϵ is such a node). By the definition of δ, the run can proceed such that all the successors $y \cdot c$ of y in T_r have $r(y \cdot c) = (x', \varphi')$ with $V_K(x') \models \varphi'$. Let $\langle T_r, r \rangle$ be a run that proceeds as above and, in addition, whenever φ is of the form $A\varphi_1 U \varphi_2$ or $E\varphi_1 U \varphi_2$ and $V_K(x) \models \varphi_2$, proceeds according to $\delta(\varphi_2, V_K(x), d(x))$. It is easy to see that all the paths in such $\langle T_r, r \rangle$ are either finite or reach a state associated with a $\tilde{U}$-formula and stay there thereafter. Thus, $\langle T_r, r \rangle$ is accepting. $\square$

As the nonemptiness problem for WAAs is in exponential time [MSS86], the above described WAAs provide also an exponential-time satisfiability procedure for CTL. In Examples 3.4 and 3.5 below, we describe the WAAs of two CTL formulas.

Example 3.4 Consider the CTL formula $\psi = A(\boldsymbol{true} U (A\boldsymbol{false}\tilde{U} p))$. Note that ψ is equivalent to the formula $AFAGp$. For every $\mathcal{D} \subset \mathbb{N}$, the WAA associated with ψ is $\mathcal{A}_{\mathcal{D},\psi} = \langle \{\{p\}, \emptyset\}, \mathcal{D}, \{\psi, A\boldsymbol{false}\tilde{U} p\}, \delta, \psi, \{A\boldsymbol{false}\tilde{U} p\} \rangle$, where δ is described in the following table (we restrict $\mathcal{A}_{\mathcal{D},\psi}$ to its reachable states).

q	$\delta(q, \{p\}, k)$	$\delta(q, \emptyset, k)$
ψ	$\bigwedge_{c=0}^{k-1}(c, A\boldsymbol{false}\tilde{U} p) \vee \bigwedge_{c=0}^{k-1}(c, \psi)$	$\bigwedge_{c=0}^{k-1}(c, \psi)$
$A\boldsymbol{false}\tilde{U} p$	$\bigwedge_{c=0}^{k-1}(c, A\boldsymbol{false}\tilde{U} p)$	$\boldsymbol{false}$

In the state ψ, if p holds in the present, $\mathcal{A}_{\mathcal{D},\psi}$ may either guess that $A\boldsymbol{false}\tilde{U} p$, the eventuality of ψ, is satisfied in the present, or proceed with $\bigwedge_{c=0}^{k-1}(c, \psi)$, which means that the requirement for fulfilling the eventuality of ψ is postponed to the future. The crucial point is that since $\psi \notin \alpha$, infinite postponing is impossible. In the state $A\boldsymbol{false}\tilde{U} p$, $\mathcal{A}_{\mathcal{D},\psi}$ expects a tree in which p is always true in all paths. Then, it keeps visiting $A\boldsymbol{false}\tilde{U} p$ forever. Since $A\boldsymbol{false}\tilde{U} p \in \alpha$, this is permitted.

Example 3.5 Consider the CTL formula $\psi = A((EX\neg p) U b)$. For every $\mathcal{D} \subset \mathbb{N}$, the WAA associated with ψ is $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{\{p,b\}}, \mathcal{D}, \{\psi, \neg p\}, \delta, \psi, \emptyset \rangle$, where δ is described in the following table (we restrict $\mathcal{A}_{\mathcal{D},\psi}$ to its reachable states).

q	$\delta(q, \{p, b\}, k)$	$\delta(q, \{p\}, k)$	$\delta(q, \{b\}, k)$	$\delta(q, \emptyset, k)$
ψ	$\boldsymbol{true}$	$\bigvee_{c=0}^{k-1}(c, \neg p) \wedge \bigwedge_{c=0}^{k-1}(c, \psi)$	$\boldsymbol{true}$	$\bigvee_{c=0}^{k-1}(c, \neg p) \wedge \bigwedge_{c=0}^{k-1}(c, \psi)$
$\neg p$	$\boldsymbol{false}$	$\boldsymbol{false}$	$\boldsymbol{true}$	$\boldsymbol{true}$

In the state ψ, if b does not hold on the present, $\mathcal{A}_{\mathcal{D},\psi}$ requires both $EX\neg p$ to be satisfied in the present (that is, $\neg p$ to be satisfied in some successor), and ψ to be satisfied by all the successors. As $\psi \notin \alpha$, $\mathcal{A}_{\mathcal{D},\psi}$ should eventually reach a node that satisfies b.

We now present a similar translation for the alternation-free μ-calculus.

- $\delta(p,\sigma,k) = \textbf{\textit{true}}$ if $p \in \sigma$. • $\delta(p,\sigma,k) = \textbf{\textit{false}}$ if $p \notin \sigma$.
- $\delta(\neg p,\sigma,k) = \textbf{\textit{true}}$ if $p \notin \sigma$. • $\delta(\neg p,\sigma,k) = \textbf{\textit{false}}$ if $p \in \sigma$.
- $\delta(\varphi_1 \wedge \varphi_2,\sigma,k) = \delta(\varphi_1,\sigma,k) \wedge \delta(\varphi_2,\sigma,k)$.
- $\delta(\varphi_1 \vee \varphi_2,\sigma,k) = \delta(\varphi_1,\sigma,k) \vee \delta(\varphi_2,\sigma,k)$.
- $\delta(AX\varphi_2,\sigma,k) = \bigwedge_{c=0}^{k-1}(c,\varphi_2)$.
- $\delta(EX\varphi_2,\sigma,k) = \bigvee_{c=0}^{k-1}(c,\varphi_2)$.
- $\delta(A\varphi_1 U\varphi_2,\sigma,k) = \delta(\varphi_2,\sigma,k) \vee (\delta(\varphi_1,\sigma,k) \wedge \bigwedge_{c=0}^{k-1}(c,A\varphi_1 U\varphi_2))$.
- $\delta(E\varphi_1 U\varphi_2,\sigma,k) = \delta(\varphi_2,\sigma,k) \vee (\delta(\varphi_1,\sigma,k) \wedge \bigvee_{c=0}^{k-1}(c,E\varphi_1 U\varphi_2))$.
- $\delta(A\varphi_1 \tilde{U}\varphi_2,\sigma,k) = \delta(\varphi_2,\sigma,k) \wedge (\delta(\varphi_1,\sigma,k) \vee \bigwedge_{c=0}^{k-1}(c,A\varphi_1 \tilde{U}\varphi_2))$.
- $\delta(E\varphi_1 \tilde{U}\varphi_2,\sigma,k) = \delta(\varphi_2,\sigma,k) \wedge (\delta(\varphi_1,\sigma,k) \vee \bigvee_{c=0}^{k-1}(c,E\varphi_1 \tilde{U}\varphi_2))$.

To show that $\mathcal{A}_{\mathcal{D},\psi}$ is a WAA, we define a partition of Q into disjoint sets and a partial order over the sets. Each formula $\varphi \in cl(\psi)$ constitutes a (singleton) set $\{\varphi\}$ in the partition. The partial order is then defined by $\{\varphi_1\} \leq \{\varphi_2\}$ iff $\varphi_1 \in cl(\varphi_2)$. Since each transition of the automaton from a state φ leads to states associated with formulas in $cl(\varphi)$, the weakness conditions hold. In particular, each set is either contained in α or disjoint from α.

Below we prove the correctness of our construction, namely, that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ contains exactly all the $\mathcal{D}$-trees that satisfy ψ. We first prove that $\mathcal{A}_{\mathcal{D},\psi}$ is sound. Given an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathcal{D},\psi}$ on a tree $\langle T_K, V_K\rangle$, we prove that for every $y \in T_r$ such that $r(y) = (x,\varphi)$, we have $V_K(x) \models \varphi$. Thus, in particular, $V_K(\epsilon) \models \psi$. The proof proceeds by induction on the structure of φ. The case where φ is an atomic proposition is immediate and the cases where φ is $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $AX\varphi_1$, or $AX\varphi_1$ follow easily, by the induction hypothesis, from the definition of δ. Less immediate are the cases where φ is an U-formula or a $\tilde{U}$-formula. Consider first the case where φ is of the form $A\varphi_1 U\varphi_2$ or $E\varphi_1 U\varphi_2$. As $\langle T_r, r\rangle$ is an accepting run, it visits the state φ only finitely often. Since $\mathcal{A}_{\mathcal{D},\psi}$ keeps inheriting φ as long as φ_2 is not satisfied, then it is guaranteed, by the definition of δ and the induction hypothesis, that along all paths or some path, as required in φ, indeed φ_2 eventually holds and φ_1 holds everywhere until then. Consider now the case where φ is of the form $A\varphi_1 \tilde{U}\varphi_2$ or $E\varphi_1 \tilde{U}\varphi_2$. Here, it is guaranteed, by the definition of δ and the induction hypothesis, that φ_2 holds either always or until both φ_2 and φ_1 hold.

We now prove that $\mathcal{A}_{\mathcal{D},\psi}$ is complete. Given a $\mathcal{D}$-tree $\langle T_K, V_K\rangle$ such that $\langle T_K, V_K\rangle \models \psi$, we prove that $\mathcal{A}_{\mathcal{D},\psi}$ accepts $\langle T_K, V_K\rangle$. Precisely, we show that there exists an accepting run of $\mathcal{A}_{\mathcal{D},\psi}$ on $\langle T_K, V_K\rangle$. Consider a run $\langle T_r, r\rangle$ and node $y \in T_r$ for which $r(y) = (x,\varphi)$ and $V_K(x) \models \varphi$

$\mathbb{N}^* \times Q_\psi$ and for every $y \in T_r$ with $r(y) = (x, w, q)$, we have $r'(y) = (x, q)$. As in the previous direction, it is easy to see that $\langle T_r, r' \rangle$ is an accepting run of $\mathcal{A}_{\mathcal{D},\psi}$ on $\langle T_K, V_K \rangle$. □

In conclusion, given an alternating automaton $\mathcal{A}_{\mathcal{D},\psi}$ such that $\mathcal{A}_{\mathcal{D},\psi}$ accepts exactly all the $\mathcal{D}$-trees that satisfy ψ, model checking of a Kripke structure K with branching degrees in $\mathcal{D}$ with respect to ψ is reducible to checking the 1-letter nonemptiness of an automaton of the same type as $\mathcal{A}_{\mathcal{D},\psi}$ and of size $O(|K| \cdot |\mathcal{A}_{\mathcal{D},\psi}|)$. By the *sufficient branching degree property* [ES84], a branching temporal logic formula ψ is satisfiable if and only if it is satisfied in an $\{n\}$-tree, where n is the number of occurrences of the path quantifier E in ψ. Hence, satisfiability of ψ can be reduced to the nonemptiness of $\mathcal{A}_{\{n\},\psi}$. In the following sections, we show how this approach can be used to derive, in a uniform way, known time complexity bounds for model checking and satisfiability of several branching temporal logics, as well as to obtain new space complexity bounds.

3.3 Primary Applications

In the previous section, we presented an automata-based method for model checking of branching temporal logics. The efficiency of our method depends on the efficiency of the translation of branching temporal logic formulas to automata as well as the efficiency of the 1-letter nonemptiness test for them. In this section we present an application of the method with respect to CTL and the alternation-free μ-calculus. For some logics, satisfactory translations and nonemptiness tests for the suitable automata are already known, yet never been used for model checking. We point on μ-calculus as such a logic.

3.3.1 Model Checking for CTL and the Alternation-free μ-Calculus

Vardi and Wolper showed how to solve the satisfiability problem for CTL via an exponential translation of CTL formulas to Büchi automata on infinite trees [VW86a]. Muller et al. provided a simpler proof, via a linear translation of branching dynamic logic formulas to WAAs [MSS88]. We extend here the ideas of Muller et al. by demonstrating a linear translation from CTL formulas to WAAs.

Theorem 3.3 *Given a CTL formula ψ and a set $\mathcal{D} \subset I\!N$, we can construct, in linear running time, a WAA $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, cl(\psi), \delta, \psi, \alpha \rangle$, such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ.*

Proof: The set α of accepting states consists of all the $\tilde{U}$-formulas in $cl(\psi)$; that is, formulas of the form $A\varphi_1 \tilde{U} \varphi_2$ or $E\varphi_1 \tilde{U} \varphi_2$. It remains to define the transition function δ. For all $\sigma \in 2^{AP}$ and $k \in \mathcal{D}$, we define:

- The acceptance condition α is defined according to the acceptance condition α_ψ of $\mathcal{A}_{\mathcal{D},\psi}$. For example, if $\alpha_\psi \subseteq Q_\psi$ is a Büchi condition, then $\alpha = W \times \alpha_\psi$ is also a Büchi condition. If $\alpha_\psi = \{\langle G_1, B_1\rangle, \ldots, \langle G_m, B_m\rangle\}$ is a Rabin condition, then $\alpha = \{\langle W \times G_1, W \times B_1\rangle, \ldots, \langle W \times G_m, W \times B_m\rangle\}$ is also a Rabin condition.

It is easy to see that $\mathcal{A}_{K,\psi}$ is of the same type as $\mathcal{A}_{\mathcal{D},\psi}$. In particular, if $\mathcal{A}_{\mathcal{D},\psi}$ is a WAA (with a partition $\{Q_1, Q_2, \ldots, Q_n\}$), then so is $\mathcal{A}_{K,\psi}$ (with a partition $\{W \times Q_1, W \times Q_2, \ldots, W \times Q_n\}$).

Proposition 3.2

(1) $|\mathcal{A}_{K,\psi}| = O(|K| \cdot |\mathcal{A}_{\mathcal{D},\psi}|)$.

(2) $\mathcal{L}(\mathcal{A}_{K,\psi})$ *is nonempty if and only if* $K \models \psi$.

Proof: (1) follows immediately from the definition of $\mathcal{A}_{K,\psi}$. Clearly, $|W \times Q_\psi| = |W| \cdot |Q_\psi|$ and $|\alpha| = |W| \cdot |\alpha_\psi|$. Also, for every $w \in W$ and for all $q \in Q_\psi$ and $\sigma \in 2^{AP}$ with $\delta_\psi(q, \sigma) \neq \textbf{\textit{false}}$, there exists a single k ($k = bd(w)$) for which $\delta(\langle w, q\rangle, \sigma, k) \neq \textbf{\textit{false}}$. Thus $|\delta| = |W| \cdot |\delta_\psi|$,

To prove (2), we show that $\mathcal{L}(\mathcal{A}_{K,\psi})$ is nonempty if and only if $\mathcal{A}_{\mathcal{D},\psi}$ accepts $\langle T_K, V_K\rangle$. Since $\mathcal{A}_{\mathcal{D},\psi}$ accepts exactly all $\mathcal{D}$-trees that satisfy ψ, and since all the branching degrees of T_K are in $\mathcal{D}$, the later holds if and only if $K \models \psi$. Given an accepting run of $\mathcal{A}_{\mathcal{D},\psi}$ on $\langle T_K, V_K\rangle$, we construct an accepting run of $\mathcal{A}_{K,\psi}$. Also, given an accepting run of $\mathcal{A}_{K,\psi}$, we construct an accepting run of $\mathcal{A}_{\mathcal{D},\psi}$ on $\langle T_K, V_K\rangle$.

Assume first that $\mathcal{A}_{\mathcal{D},\psi}$ accepts $\langle T_K, V_K\rangle$. Thus, there exists an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathcal{D},\psi}$ on $\langle T_K, V_K\rangle$. Recall that T_r is labeled with $\mathbb{N}^* \times Q_\psi$. A node $y \in T_r$ with $r(y) = (x, q)$ corresponds to a copy of $\mathcal{A}_{\mathcal{D},\psi}$ that is in the state q and reads the tree obtained from unwinding K from $V_K(x)$. Consider the tree $\langle T_r, r'\rangle$ where T_r is labeled with $\mathbb{N}^* \times W \times Q_\psi$ and for every $y \in T_r$ with $r(y) = (x, q)$, we have that $r'(y) = (x, V_K(x), q)$. We show that $\langle T_r, r'\rangle$ is an accepting run of $\mathcal{A}_{K,\psi}$. In fact, since α is defined according to α_ψ by crossing it with W, we only need to show that $\langle T_r, r'\rangle$ is a run of $\mathcal{A}_{K,\psi}$; acceptance follows from being $\langle T_r, r\rangle$ accepting. Intuitively, $\langle T_r, r'\rangle$ is a "legal" run, since the W-component in r' always agrees with V_K. This agreement is the only additional requirement of δ with respect to δ_ψ.

Consider a node $y \in T_r$ with $r(y) = (x, q)$, $V_K(x) = w$, and $succ_R(w) = \langle w_0, \ldots, w_{k-1}\rangle$. Let $\delta_\psi(q, w, k) = \theta$. Since $\langle T_r, r\rangle$ is a run of $\mathcal{A}_{\mathcal{D},\psi}$, there exists a set $S = \{(c_0, q_0), (c_1, q_1), \ldots, (c_n, q_n)\}$, such that S satisfies θ, and the successors of y in T_r are $y \cdot i$, for $1 \leq i \leq n$, each labeled with $(x \cdot c_i, q_i)$. In $\langle T_r, r'\rangle$, by its definition, $r'(y) = (x, w, q)$ and the successors of y are $y \cdot i$, each labeled with $(x \cdot c_i, w_{c_i}, q_i)$. Let $\delta(q, a, bd(w)) = \theta'$. Hence, by the definition of δ, the set $\{(c_0, w_{c_0}, q_0), (c_1, w_{c_1}, q_1), \ldots, (c_n, w_{c_n}, q_n)\}$, satisfies θ'. Thus, $\langle T_r, r'\rangle$ is a run of $\mathcal{A}_{K,\psi}$.

Assume now that $\mathcal{A}_{K,\psi}$ accepts some tree. Thus, there exists an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{K,\psi}$. Also, T_r is labeled with $\mathbb{N}^* \times W \times Q_\psi$. Consider the tree $\langle T_r, r'\rangle$ where T_r is labeled with

Augmenting Q with an accepting sink q_{acc}, we can therefore define δ' as a mapping into 2^{Q^n}. For each state q, we define $\delta'(q,a,n)$ as follows.

- If $inflate(\delta(q,a)) = \mathbf{true}$, then $\delta'(q,a,n) = \{\langle q_{acc}, q_{acc}, \ldots, q_{acc}\rangle\}$.
- If $inflate(\delta(q,a)) = \mathbf{false}$, then $\delta'(q,a,n) = \emptyset$.
- If $inflate(\delta(q,a)) = (c_1, q_{c_1}) \wedge (c_2, q_{c_2})$, then $\delta'(q,a,n) = \{\langle s_0, s_1, \ldots, s_{n-1}\rangle\}$, where $s_{c_1} = q_{c_1}$, $s_{c_2} = q_{c_2}$, and $s_i = q_{acc}$ for $i \notin \{c_1, c_2\}$.
- If $inflate(\delta(q,a)) = (c_1, q_{c_1}) \vee (c_2, q_{c_2})$, then $\delta'(q,a,n) = \{\langle s^1_0, s^1_1, \ldots, s^1_{n-1}\rangle, \langle s^2_0, s^2_1, \ldots, s^2_{n-1}\rangle\}$, where $s^1_{c_1} = q_{c_1}$, $s^2_{c_2} = q_{c_2}$, and $s^j_i = q_{acc}$ for $(j,i) \notin \{(1,c_1),(2,c_2)\}$.

The resulted automaton is a nondeterministic tree automaton.

We show that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Assume first that $\mathcal{L}(\mathcal{A}) \neq \emptyset$. Then, there exists an accepting run $\langle T, r\rangle$ of $\mathcal{A}$ over a^ω. It is easy to see that the tree $\langle T, r'\rangle$, in which $r'(\epsilon) = r(\epsilon)$ and for all $y \cdot c \in T$ with $r'(y) = (x,q)$ and $r(y \cdot c) = (0^j, q_i)$, we have $r'(y \cdot c) = (x \cdot i, q_i)$, is an accepting run of $\mathcal{A}'$ over the a-labeled tree with a fixed branching degree $|Q|$.

Assume first that $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Then, there exists an accepting run $\langle T, r'\rangle$ of $\mathcal{A}'$ over the a-labeled tree with a fixed degree $|Q|$. It is easy to see that the tree $\langle T, r\rangle$, in which for all $y \in T$ with $r(y) = (x,q)$ we have $r'(y) = (0^{|x|}, q)$, is an accepting run of $\mathcal{A}$ over a^ω. It is also easy to see that the type of $\mathcal{A}$ is preserved. □

We will study later the complexity of the 1-letter nonemptiness problem.

3.2.2 The Product Automaton

As discussed above, the nonemptiness problem for alternating tree automata can not, in general, be reduced to the 1-letter nonemptiness problem. It is taking the product with K that makes the reduction valid here and yields an automaton over a 1-letter alphabet. Since each state in this product expects a single input, it is guaranteed that all the copies that run on the same subtree guess the same Σ-labeling: the one corresponding to the unwinding of K.

Let $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, Q_\psi, \delta_\psi, Q^0, \alpha_\psi\rangle$ be an alternating automaton which accepts exactly all the $\mathcal{D}$-trees that satisfy ψ and let $K = \langle AP, W, R, w^0, L\rangle$ be a Kripke structure with branching degrees in $\mathcal{D}$. The product automaton of $\mathcal{A}_{\mathcal{D},\psi}$ and K is $\mathcal{A}_{K,\psi} = \langle \{a\}, \mathcal{D}, W \times Q_\psi, \delta, w^0 \times Q^0, \alpha\rangle$ where δ and α are defined as follows:

- Let $q \in Q_\psi$, $w \in W$, $succ_R(w) = \langle w_0, \ldots, w_{bd(w)-1}\rangle$, and $\delta_\psi(q, L(w), bd(w)) = \theta$. Then $\delta(\langle w,q\rangle, a, bd(w)) = \theta'$, where θ' is obtained from θ by replacing each atom (c,q') in θ by the atom $(c, \langle w_c, q'\rangle)$. For $k \neq bd(w)$, we have $\delta(\langle w,q\rangle, a, k) = \mathbf{false}$.

It is easy to see that all the transitions in δ' are simple.

Given a set $S \subseteq Q$, let $[S] = \{[q] : q \in S\}$. Consider the alphabets $\Sigma = \mathbb{N}^* \times [Q]$ and $\Sigma' = \mathbb{N}^* \times Q'$, and consider an infinite word b' over Σ'. The (possibly finite) word $b'_{|\Sigma}$ is obtained from b' by restricting it to its letters in Σ. We say that a word b' over Σ' *corresponds* to a word b over Σ iff $b'_{|\Sigma} = b$.

Consider a run $\langle T', r' \rangle$ of $\mathcal{A}'$. For every $F \subseteq Q$ and for every path $\rho' \subseteq T'$, we have that ρ' visits infinitely many states in $[F]$ iff $\rho'_{|\Sigma}$ visits infinitely many states in $[F]$. Accordingly, as all our acceptance conditions only refer to visiting subsets of Q infinitely often, the definition of F' ignores states which are not of the form $[q]$. Thus, if F is a Büchi acceptance condition, then $F' = [F]$, and if $F = \{\langle G_1, B_1 \rangle, \ldots, \langle G_m, B_m \rangle\}$ is a Rabin or Streett acceptance condition, then $F' = \{\langle [G_1], [B_1] \rangle, \ldots, \langle [G_m], [B_m] \rangle\}$.

Each run $\langle T, r \rangle$ of $\mathcal{A}$ corresponds to a run $\langle T', r' \rangle$ of $\mathcal{A}'$, in the sense that for each path $\rho \subseteq T$ there exists a path $\rho' \subseteq T'$ such that $r'(\rho')$ corresponds to $r(\rho)$, and vice versa. Indeed, the run $\langle T', r' \rangle$ proceeds exactly as $\langle T, r \rangle$, only in "smaller steps". Similarly, each run $\langle T', r' \rangle$ of $\mathcal{A}'$ corresponds to a run $\langle T, r \rangle$ of $\mathcal{A}$. Here, $\langle T, r \rangle$ proceeds in larger steps. By the definition of δ, it is guaranteed that when necessary, $\mathcal{A}$ can cluster several transitions of $\mathcal{A}'$ into a single transition. By the definition of F', we have that $\langle T, r \rangle$ is accepting iff $\langle T', r' \rangle$ is accepting. Thus, $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$.

We prove that $\mathcal{A}$ and $\mathcal{A}'$ are of the same type. Clearly, they agree on the type of their acceptance conditions. We show that if $\mathcal{A}$ is weak then so is $\mathcal{A}'$. Let $\{Q_1, Q_2, \ldots, Q_n\}$ be the partition of Q into sets such that $Q_1 \leq \ldots \leq Q_n$ is an extension of the partial order to a total order. We define a partition $\{Q'_1, Q'_2, \ldots, Q'_n\}$ of Q' as follows. For each state $q \in Q$, let $add(q)$ be the set of all subformulas of $\delta(q, a)$ (including $\delta(q, a)$ itself). For a set Q_i, let $add(Q_i) = \bigcup_{q \in Q_i} add(q)$.

1. $[Q_i] \subseteq Q'_i$, and
2. For each $q' \notin [Q]$, we have $q' \in Q'_i$ such that $i = \min\{j : q' \in [add(Q_j)]\}$. That is, a state associated with a subformula θ belongs to the least Q_i from which θ is reachable.

It is easy to see that $\{Q'_1, Q'_2, \ldots, Q'_n\}$ is a valid partition, with the ordering $Q'_1 \leq \ldots \leq Q'_n$.

It is left to reduce problem 3 to problem 4. Let $\mathcal{A} = \langle \{a\}, Q, \delta, q_0, F \rangle$ be a simple alternating word automaton. Consider the alternating tree automaton $\mathcal{A}' = \langle \{a\}, \{n\}, Q, \delta', q_0, F \rangle$, where $n = |Q|$ and for all $q \in Q$ we have that $\delta'(q, a, n) = inflate(\delta(q, a))$. The fact that the transitions in δ are simple guarantees that so are the transitions in δ'. Also, the definition of $inflate$ guarantees that for all $q \in Q$ and all atoms (i, q_j) in $\delta'(q, a, n)$, we have that $i = j$. Thus, if $\delta'(q, a, n)$ is a conjunction which sends two copies of the automaton to the same direction, then these copies enter the same state and they are equivalent to a single copy sent to this direction.

- $deflate((0,q_0) \wedge (0,q_2) \vee (0,q_2) \wedge (1,q_2) \wedge (1,q_1)) = q_0 \wedge q_2 \vee q_2 \wedge q_2 \wedge q_1$.
- $inflate(q_0 \wedge q_2 \vee q_2 \wedge q_2 \wedge q_1) = (0,q_0) \wedge (2,q_2) \vee (2,q_2) \wedge (2,q_2) \wedge (1,q_1))$.

We start by reducing problem 1 to problem 2. Given an alternating tree automaton $\mathcal{A} = \langle \{a\}, \mathcal{D}, Q, \delta, q_0, F \rangle$, consider the alternating word automaton $\mathcal{A}' = \langle \{a\}, Q, \delta', q_0, F \rangle$, where for all $q \in Q$, we have

$$\delta'(q,a) = \bigvee_{k \in \mathcal{D}} deflate(\delta(q,a,k)).$$

We show that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Intuitively, since the 1-letter tree is homogeneous (i.e., all its subtrees are the same), its branching structure is not important.

Assume first that $\mathcal{L}(\mathcal{A}) \neq \emptyset$. Then, there exists an accepting run $\langle T, r \rangle$ of $\mathcal{A}$ over an a-labeled tree with degrees in $\mathcal{D}$. It is easy to see that the tree $\langle T, r' \rangle$, in which for all $y \in T$ with $r(y) = (x,q)$ we have $r'(y) = (0^{|x|}, q)$, is an accepting run of $\mathcal{A}'$ over a^ω (recall that one can regard a run of an alternating word automaton as a run of an alternating tree automaton with $\mathcal{D} = \{1\}$). In a similar (though more tedious) way, an accepting run of $\mathcal{A}'$ induces an accepting run of $\mathcal{A}$. Clearly, the reduction preserves the type of $\mathcal{A}$.

We now reduce problem 2 to problem 3. Given an alternating word automaton $\mathcal{A} = \langle \{a\}, Q, \delta, q_0, F \rangle$, we define an alternating word automaton $\mathcal{A}' = \langle \{a\}, Q', \delta', q_0, F' \rangle$, such that for all $q \in Q'$, the formula $\delta'(q,a)$ is simple, and $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$. The idea is that since the 1-letter tree is homogeneous, we can regard transitions of $\mathcal{A}'$ as ϵ-transitions and replace a transition in δ with several simple transitions in δ'. This requires an extension of the state set of $\mathcal{A}$ in at most the number of subformulas of transitions in δ. We assume, without loss of generality, that for all $q \in Q$, the transition $\delta(q,a)$ is either ***true***, ***false***, or has the form $\theta_1 * \theta_2$ ($*$ denotes $\wedge$ or $\vee$). Describing $\mathcal{A}'$, we denote a state in Q' as $[\theta]$, where $\theta \in \mathcal{B}^+(Q)$. We define Q' inductively as follows:

- For each $q \in Q$, we have $[\delta(q,a)] \in Q'$.
- For each $[\theta_1 * \theta_2] \in Q'$, we have $[\theta_1] \in Q'$ and $[\theta_2] \in Q'$.

Thus, a state in Q' is either $[\boldsymbol{true}]$, $[\boldsymbol{false}]$, $[\theta_1 * \theta_2]$, or $[q]$, for $q \in Q$. We define δ' as follows:

- $\delta'([\boldsymbol{true}], a) = \boldsymbol{true}$.
- $\delta'([\boldsymbol{false}], a) = \boldsymbol{false}$.
- $\delta'([\theta_1 * \theta_2], a) = [\theta_1] * [\theta_2]$.
- $\delta'([q], a) = \delta'([\delta(q,a)], a)$.

3.2.1 The 1-letter Nonemptiness Problem

The nonemptiness problem for nondeterministic tree automata is reducible to the 1-letter nonemptiness problem for them [Rab69]. Precisely, instead checking the nonemptiness of an automaton $\mathcal{A} = \langle \Sigma, Q, \delta, Q^0, \alpha \rangle$, one can check the nonemptiness of the automaton $\mathcal{A}' = \langle \{a\}, Q, \delta', Q^0, \alpha \rangle$ where for all $q \in Q$, we have $\delta'(q,a) = \bigcup_{\sigma \in \Sigma} \delta(q,\sigma)$. It is easy to see that if $\mathcal{A}$ accepts some tree, then $\mathcal{A}'$ accepts the a-labeled tree. Also, as each transition of $\mathcal{A}'$ originates from a transition of $\mathcal{A}$, it is not hard to see that if $\mathcal{A}'$ accepts the a-labeled tree, then there exists a tree that $\mathcal{A}$ accepts. This can be viewed as if $\mathcal{A}'$ first guesses a Σ-labeling for the input tree and then proceeds like $\mathcal{A}$ on this Σ-labeled tree. This reduction is not valid for alternating tree automata! There, if $\mathcal{A}'$ accepts the a-labeled tree, it is still not guaranteed that $\mathcal{A}$ accepts some tree. A necessary condition for the validity of the reduction is that different copies of $\mathcal{A}'$ that run on the same subtree guess the same Σ-labeling for this subtree. Nothing, however, prevents one copy of $\mathcal{A}'$ to proceed according to one labeling and another copy to proceed according to a different labeling. This problem does not occur when $\mathcal{A}$ is defined over a singleton alphabet. There, it is guaranteed that all copies proceed according to the same (single) labeling. The theorem below considers the complexity of the 1-letter nonemptiness problem.

Theorem 3.1 *For automata of a certain type, the following four problems are intereducible in both linear time and logarithmic space.*

1. *1-letter nonemptiness of alternating tree automata.*
2. *1-letter nonemptiness of alternating word automata.*
3. *1-letter nonemptiness of simple alternating word automata.*
4. *Nonemptiness of nondeterministic tree automata.*

Proof: The nonemptiness problem for nondeterministic tree automata is reducible to the 1-letter nonemptiness problem for them [Rab69]. In addition, the reduction preserves the type of the automaton. Hence, as nondeterministic tree automata are a special case of alternating tree automata, a reduction from problem 4 to problem 1 is straightforward. So, we only have to show reductions from problem 1 to problem 2, from problem 2 to problem 3, and from problem 3 to problem 4. Before we get to the proof we define the two functions $deflate : \mathcal{B}^+(\mathbb{N} \times Q) \to \mathcal{B}^+(Q)$ and $inflate : \mathcal{B}^+(Q) \to \mathcal{B}^+(\mathbb{N} \times Q)$. For a formula $\theta \in \mathcal{B}^+(\mathbb{N} \times Q)$, the formula $deflate(\theta)$ is the formula obtained from θ by replacing each atom (c,q) in θ by the atom q. The function $inflate$ is defined with respect to an enumeration $q_0, q_1, \ldots, q_n$ of the states in Q. For a formula $\theta \in \mathcal{B}^+(Q)$, the formula $inflate(\theta)$ is the formula in $\mathcal{B}^+(\{0, \ldots n\} \times Q)$ obtained from θ by replacing each atom q_c in θ by the atom (c, q_c). For example,

many computations. Model checking is thus reduced to checking inclusion between the set of computations allowed by the Kripke structure and the language of an automaton describing the formula [VW86a]. For branching temporal logic, each Kripke structure corresponds to a single nondeterministic computation. On that account, model checking is reduced to checking the membership of this computation in the language of the automaton describing the formula [Wol89]. We show here that alternating automata are the suitable framework for automata-based model-checking algorithms.

A Kripke structure $K = \langle AP, W, R, w^0, L\rangle$ can be viewed as a W-labeled tree $\langle T_K, V_K\rangle$ that corresponds to the unwinding of K from w^0. Formally, for every node w, let $succ_R(w) = \langle w_0, \ldots, w_{bd(w)-1}\rangle$ be an ordered list of w's R-successors (we assume that the nodes of W are ordered). We define T_K and V_K as the smallest sets that satisfy the following:

1. $\epsilon \in T_K$ and $V_K(\epsilon) = w^0$.
2. For $y \in T_K$ with $succ_R(V_K(y)) = \langle w_0, \ldots, w_m\rangle$ and for $0 \leq i \leq m$, we have $y \cdot i \in T_K$ and $V_K(y \cdot i) = w_i$.

We sometimes refer to $\langle T_K, V_K\rangle$ as a computation tree, taking instead of $V_K(x)$, the label $L(V_K(x))$. It will be clear from the context which interpretation we refer to.

Let ψ be a branching temporal formula and let $\mathcal{D} \subset \mathbb{N}$ be a set of branching degrees. Suppose that $\mathcal{A}_{\mathcal{D},\psi}$ is an alternating automaton that accepts exactly all the $\mathcal{D}$-trees that satisfy ψ (the details of the construction of $\mathcal{A}_{\mathcal{D},\psi}$ depend on the logic in which ψ is specified; we consider some examples in the next sections). Consider a product of K and $\mathcal{A}_{\mathcal{D},\psi}$; i.e., an automaton that accepts the language $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi}) \cap \{\langle T_K, V_K\rangle\}$. The language of this product either contains a single tree, $\langle T_K, V_K\rangle$, in which case $K \models \psi$, or is empty, in which case $K \not\models \psi$. This discussion suggests the following automata-based model-checking algorithm. Given a branching temporal formula ψ and a Kripke structure K with branching degrees in $\mathcal{D}$, proceed as follows.

(1) Construct the alternating automaton $\mathcal{A}_{\mathcal{D},\psi}$.

(2) Construct an alternating automaton $\mathcal{A}_{K,\psi} = K \times \mathcal{A}_{\mathcal{D},\psi}$ by taking the product of K and $\mathcal{A}_{\mathcal{D},\psi}$. This automaton simulates a run of $\mathcal{A}_{\mathcal{D},\psi}$ on $\langle T_K, V_K\rangle$.

(3) Output "Yes" if $\mathcal{L}(\mathcal{A}_{K,\psi}) \neq \emptyset$, and "No", otherwise.

The type of $\mathcal{A}_{\mathcal{D},\psi}$ and consequently, the type of $\mathcal{A}_{K,\psi}$ as well as the complexity of its nonemptiness test, depend on the logic in which ψ is specified. The crucial point in our approach is that the automaton $\mathcal{A}_{K,\psi}$ is an automaton over a 1-letter alphabet; this reduces the complexity of the nonemptiness test. In the remainder of this section we discuss the 1-letter nonemptiness problem for alternating tree automata and present the product automaton $\mathcal{A}_{K,\psi}$.

of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T; in contrast, in a run of a nondeterministic automaton on $\langle T, V \rangle$ there is a one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its successors have to satisfy the transition function. Formally, $\langle T_r, r \rangle$ is a Σ_r-labeled tree where $\Sigma_r = \mathbb{N}^* \times Q$ and $\langle T_r, r \rangle$ satisfies the following:

1. $\epsilon \in T_r$ and $r(\epsilon) = (\epsilon, q_0)$, for $q_0 \in Q^0$.
2. Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x), d(x)) = \theta$. Then there is a possibly empty set $S = \{(c_0, q_0), (c_1, q_1), \ldots, (c_n, q_n)\} \subseteq \{0, \ldots, d(x) - 1\} \times Q$, such that the following hold:
   - S satisfies θ, and
   - for all $0 \leq i \leq n$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = (x \cdot c_i, q_i)$.

For example, if $\langle T, V \rangle$ is a binary tree with $V(\epsilon) = a$, $Q^0 = \{q_0\}$, and $\delta(q_0, a) = ((0, q_1) \vee (0, q_2)) \wedge ((0, q_3) \vee (1, q_2))$, then the nodes of $\langle T_r, r \rangle$ at level 1 include the label $(0, q_1)$ or $(0, q_2)$, and include the label $(0, q_3)$ or $(1, q_2)$. Note that if $\theta = \boldsymbol{true}$, then y need not have successors. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta = \boldsymbol{false}$, we can not have a run with $\theta = \boldsymbol{false}$. A run $\langle T_r, r \rangle$ is accepting iff all its infinite paths satisfy the acceptance condition. As with nondeterministic automata, an automaton accepts a tree iff there exists a run that accepts it. We denote by $\mathcal{L}(\mathcal{A})$ the language of the automaton $\mathcal{A}$; i.e., the set of all Σ-labeled trees that $\mathcal{A}$ accepts.

In [MSS86], Muller et al. introduce *weak alternating automata* (WAAs). In a WAA, the acceptance condition is $\alpha \subseteq Q$ and there exists a partition of Q into disjoint sets, Q_i, such that for each set Q_i, either $Q_i \subseteq \alpha$, in which case Q_i is an *accepting set*, or $Q_i \cap \alpha = \emptyset$, in which case Q_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the Q_i's such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma, k)$, for some $\sigma \in \Sigma$ and $k \in \mathcal{D}$, we have $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. It follows that every infinite path of a run of a WAA ultimately gets "trapped" within some Q_i. The path then satisfies the acceptance condition if and only if Q_i is an accepting set. Note that this corresponds to the Büchi acceptance condition. Indeed, a run visits infinitely many states in α iff it gets trapped in an accepting set. We sometimes refer to the *type* of an automaton, meaning its acceptance condition and its being weak or not weak.

3.2 The Method

In this section we introduce an automata-theoretic approach to model checking for branching temporal logic. For linear temporal logic, each Kripke structure may correspond to infinitely

to the node $x \cdot 1$ (the right successor of x).

For a given set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas ***true*** and ***false*** and, as usual, $\wedge$ has precedence over $\vee$. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning ***true*** to elements in Y and assigning ***false*** to elements in $X \setminus Y$ satisfies θ. We can represent δ using $\mathcal{B}^+(\{0,1\} \times Q)$. For example, $\delta(q,\sigma) = \{\langle q_1, q_2\rangle, \langle q_3, q_1\rangle\}$ can be written as $\delta(q,\sigma) = (0,q_1) \wedge (1,q_2) \vee (0,q_3) \wedge (1,q_1)$, meaning that the automaton can choose between two splitting possibilities. In the first, the copy that proceeds to direction 0 enters the state q_1 and the one that proceeds to direction 1 enters the state q_2. In the second, the copy that proceeds to direction 0 enters the state q_3 and the one that proceeds to direction 1 enters the state q_1.

In nondeterministic tree automata, each conjunction in δ has exactly one element associated with each direction. In alternating automata on binary trees, $\delta(q,\sigma)$ can be an arbitrary formula from $\mathcal{B}^+(\{0,1\} \times Q)$. We can have, for instance, a transition

$$\delta(q,\sigma) = (0,q_1) \wedge (0,q_2) \vee (0,q_2) \wedge (1,q_2) \wedge (1,q_3).$$

The above transition illustrates that several copies may go to the same direction and that the automaton is not required to send copies to all the directions. Formally, a finite alternating automaton on infinite binary trees is a tuple $\mathcal{A} = \langle \Sigma, \{2\}, Q, \delta, Q^0, \alpha\rangle$ where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(\{0,1\} \times Q)$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and α specifies the acceptance condition.

Generalizing alternating automata to trees where nodes can have different branching degrees, we have that the transition function is $\delta : Q \times \Sigma \times \mathcal{D} \to \mathcal{B}^+(\mathbb{N} \times Q)$ with the requirement that for every $k \in \mathcal{D}$, we have $\delta(q,\sigma,k) \in \mathcal{B}^+(\{0,\ldots,k-1\} \times Q)$. When the automaton is in a state q as it reads a node that is labeled by a letter σ and has k successors, it applies the transition $\delta(q,\sigma,k)$. For each $q \in Q$ and $\sigma \in \Sigma$, we denote $\bigvee_{k \in \mathcal{D}} \delta(q,\sigma,k)$ by $\delta(q,\sigma)$. As with nondeterministic automata, we define the *size of an alternating automaton* as $|Q| + |F| + |\delta|$. Here, however, $|\delta|$ is the sum of lengths of formulas that appear as $\delta(q,\sigma,k)$ for some $q \in Q$, $\sigma \in \Sigma$, and $k \in \mathcal{D}$, and we take the length of the formulas ***true*** and ***false*** as 0. Note that $\mathcal{A}$ can be stored in space $O(|\mathcal{A}|)$.

A formula in $\mathcal{B}^+(X)$ is *simple* if it is of the form $X * X$, where $* \in \{\wedge, \vee\}$. An alternating automaton is simple if all its transitions are simple. Thus, in each transition, a simple alternating automaton splits into two copies in either a universal or an existential mode.

A *run of an alternating automaton* $\mathcal{A}$ on a tree $\langle T, V\rangle$ is a tree $\langle T_r, r\rangle$ in which the root is labeled by $q_0 \in Q^0$ and every other node is labeled by an element of $\mathbb{N}^* \times Q$. Unlike T, in which each node has at least one successor, the tree T_r may have *leaves* (nodes with no successors). Thus, a path in T_r may be either finite, in which case it contains a leaf, or infinite. Each node

tree. We showed that CTL model checking is *linearly* reducible to acceptance of a simultaneous tree by an automaton. Precisely, with every CTL formula ψ, we associated an automaton $\mathcal{A}_\psi$ that accepts a simultaneous tree iff it originates from a Kripke structure satisfying ψ. Model checking of ψ in a Kripke structure K is then reduced to the membership of the simultaneous tree induced by K in the language of $\mathcal{A}_\psi$. The later can be checked in linear time, matching the known bound for CTL model checking. Satisfiability of ψ, however, can not be reduced to the nonemptiness problem of the language of $\mathcal{A}_\psi$. Indeed, acceptance of a non-simultaneous tree by $\mathcal{A}_\psi$ has nothing to do with the satisfiability of ψ. Instead, satisfiability of ψ should be reduced to the nonemptiness problem of the language of $\mathcal{A}_\psi$ with respect to simultaneous trees. This problem is much harder. It involves an exponential blow up in the size of $\mathcal{A}_\psi$, matching the exponential gap between model checking and satisfiability of CTL.

Simultaneous trees and the framework suggested in [BG93] were a first and meaningful evidence to the possibility of applying automata-theoretic techniques to branching-time model checking. Still, they suffered from an ad-hoc approach and we find their main importance in being a fountainhead to the better framework and to its many applications that we describe in the rest of this chapter. For this reason we prefered to omit here the technical details of simultaneous trees and for this reason we will always have a warm place in our hearts for them.

In [BVW94] we suggested a second solution to the problem. The solution is based on the conjecture that there should be automata for which the membership problem is easier than the nonemptiness problem. Having such automata, we will be able to translate branching temporal logic formulas into succinct automata for which the membership problem is cheap, matching model checking complexity, and for which the nonemptiness problem is expensive, matching satisfiability complexity. The conjecture turned out to be true. Today we know that alternating tree automata satisfy the above qualifications and that they are the key to a comprehensive automata-theoretic framework for branching temporal logics.

3.1 Alternating Tree Automata

Alternation is a generalization of nondeterminism [CKS81]. *Alternating tree automata* generalize nondeterministic tree automata and were first introduced in [MS87]. For simplicity, we refer first to automata on binary trees (i.e., when $T = \{0,1\}^*$ and $\mathcal{D} = \{2\}$). Consider a nondeterministic tree automaton $\mathcal{A} = \langle \Sigma, \{2\}, Q, \delta, Q^0, \alpha \rangle$. The transition relation δ maps an automaton state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a set of pairs of states. Each such pair suggests a nondeterministic choice for the automaton's next configuration. When the automaton is in a state q as it reads a node x labeled by a letter σ, it proceeds by first choosing a pair $\langle q_1, q_2 \rangle \in \delta(q, \sigma)$, and then splitting into two copies. One copy enters the state q_1 and proceeds to the node $x \cdot 0$ (the left successor of x), and the other copy enters the state q_2 and proceeds

Chapter 3

An Automata-Theoretic Approach to Branching-Time Model Checking

Traditional automata-theoretic methods for temporal logics translate a temporal logic formula ψ into a nondeterministic automaton $\mathcal{A}_\psi$ that accepts exactly all the models of ψ. Doing so, satisfiability of ψ is reduced to the nonemptiness of $\mathcal{A}_\psi$ and model checking of ψ in a Kripke structure K is reduced either to an inclusion checking between the computations of K and the language of $\mathcal{A}_\psi$ (in case ψ is a linear temporal logic formula) or to a membership checking of the tree obtained by unwinding K in the language of $\mathcal{A}_\psi$ (in case ψ is a branching temporal logic). Thus, both model checking and satisfiability are handled by the same automaton. The size of $\mathcal{A}_\psi$ is therefore a crucial factor in the complexity of these problems. For example, we can not expect a linear-time model-checking algorithm that is based on an exponential-size $\mathcal{A}_\psi$. For nondeterministic automata, the complexity of the nonemptiness problem, the membership problem, and the inclusion problem coincide. While this is not an obstacle for linear temporal logics, where the complexities of model checking and satisfiability coincide too, this is an obstacle for branching temporal logics. Here, model checking is typically cheaper than satisfiability.

In [BG93], we suggested a first solution to this problem. The solution is based on the idea that the compulsion of having the same automaton for model checking and satisfiability is not an indispensable one. For model checking, $\mathcal{A}_\psi$ does not have to accepts exactly all the trees that originate from unwinding Kripke structures that satisfy ψ. Instead, $\mathcal{A}_\psi$ can be defined to accept some mutation of these trees; a mutation which, on the one hand, can reduce the state space of $\mathcal{A}_\psi$ and, on the other hand, is easily obtained from the structure K. *Simultaneous trees* are such a mutation. Simultaneous trees are trees in which each sub-tree is duplicated twice as the two leftmost successors of its root. This enables a tree automaton to visit different nodes of the same path simultaneously. Consequently, an automaton that assumes simultaneous input trees is more succinct than an automaton that handles nonrestricted trees. In addition, unwinding a Kripke structure into a simultaneous tree is as easy as unwinding it to a usual computation

set states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ, the transition function of a tree automaton maps a state, a letter, and a branching degree k, to the set of k-tuples of states, each suggesting the automaton k states (one for each successor) to move into.

A run of $\mathcal{A}$ on an input Σ-labeled tree $\langle T, V\rangle$ is a Q-labeled tree $\langle T, r\rangle$ such that $r(\epsilon) \in Q^0$ and for every $x \in T$, we have that $\langle r(x \cdot 0), r(x \cdot 1), \ldots, r(x \cdot (d(x) - 1))\rangle \in \delta(r(x), V(x), d(x))$. If, for instance, $r(0) = q$, $V(0) = \sigma$, $d(0) = 2$, and $\delta(q, \sigma, 2) = \{\langle q_1, q_2\rangle, \langle q_4, q_5\rangle\}$, then either $r(0 \cdot 0) = q_1$ and $r(0 \cdot 1) = q_2$, or $r(0 \cdot 0) = q_4$ and $r(0 \cdot 1) = q_5$. Given a run $\langle T, r\rangle$ and a path $\rho \subseteq T$, we define

$$Inf(r|\rho) = \{q \in Q \ : \ \text{for infinitely many } x \in \rho, \text{ we have } r(x) = q\}.$$

Acceptance of a Σ-labeled tree $\langle T, V\rangle$ by a run r of a tree automaton is defined, according to α, with respect to $Inf(r|\rho)$ of all paths $\rho \subseteq T$. As in the case of words, $Inf(r|\rho)$ can either satisfy or not satisfy the acceptance condition. A run $\langle T, r\rangle$ then accepts $\langle T, V\rangle$ iff $Inf(r|\rho)$ satisfies the acceptance condition for all paths $\rho \subseteq T$. An automaton $\mathcal{A}$ accepts $\langle T, V\rangle$ iff there exists a run $\langle T, r\rangle$ of $\mathcal{A}$ on $\langle T, V\rangle$ such that $\langle T, r\rangle$ accepts $\langle T, V\rangle$. As with word automata, $\mathcal{L}(\mathcal{A})$ denotes the language of $\mathcal{A}$.

Note that a word automaton can be viewed as a tree automaton with $\mathcal{D} = \{1\}$. We define the *size of an automaton* $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, \delta, Q^0, \alpha\rangle$ as $|Q| + |\alpha| + |\delta|$, where $|\delta|$ is the sum of lengths of tuples that appear as $\delta(q, \sigma, k)$ for some $q \in Q$, $\sigma \in \Sigma$, and $k \in \mathcal{D}$. We denote the size of $\mathcal{A}$ by $|\mathcal{A}|$. Note that $\mathcal{A}$ can be stored in space $O(|\mathcal{A}|)$.

Of special interest to us is the problem of deciding whether the language of a given automaton is nonempty; i.e., given an automaton $\mathcal{A}$, testing whether there exists an object that $\mathcal{A}$ accepts. The following complexity results are known.

Theorem 2.2

(1) [EL85a, EL85b] The nonemptiness problem for Büchi word automata can be solved in linear time.

(2) [VW94] The nonemptiness problem for Büchi word automata can be solved in NLOGSPACE.

(3) [VW86a] The nonemptiness problem for Büchi tree automata can be solved in quadratic time.

(4) [Eme85, VS85, EJ88] The nonemptiness problem for Rabin tree automata is NP-complete.

a run, are as in the case of finite automata on finite words. The last component, α, specifies the *acceptance condition*, namely, a condition that $Inf(r)$ should satisfy in order for r to accepts w. Several acceptance conditions, inducing several types of automata, are studied in the literature. We mention here three:

- *Büchi automata*, where $\alpha \subseteq Q$, and r accepts w iff $Inf(r) \cap \alpha \neq \emptyset$.
- *Rabin automata*, where $\alpha \subseteq 2^Q \times 2^Q$, and r accepts w iff there exists a pair $\langle G, B \rangle \in \alpha$, such that $Inf(r) \cap G \neq \emptyset$ and $Inf(r) \cap B = \emptyset$.
- *Streett automata*, where $\alpha \subseteq 2^Q \times 2^Q$, and r accepts w iff for every pair $\langle G, B \rangle \in \alpha$, we have that $Inf(r) \cap G = \emptyset$ or $Inf(r) \cap B \neq \emptyset$.

The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of infinite words that $\mathcal{A}$ accepts. Thus, each word automaton defines a subset of Σ^ω. Below we define trees and automata that run on infinite trees.

A *tree* is a set $T \subseteq \mathbb{N}^*$ such that if $x \cdot c \in T$ where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $x \in T$, and for all $0 \leq c' < c$, we have that $x \cdot c' \in T$. The elements of T are called *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot c$ where $c \in \mathbb{N}$ are the *successors* of x. The number of successors of a node x is called the degree of x and is denoted $d(x)$. We consider here trees in which each node has at least one successor. A *path* ρ of a tree T is a set $\rho \subseteq T$ such that $\epsilon \in \rho$ and for every $x \in \rho$ there exists a unique $c \in \mathbb{N}$ such that $x \cdot c \in \rho$. For a path ρ and $j \geq 0$, let ρ_j denote the node of length j in ρ. Intuitively, each path $\rho \subseteq T$ induces a unique sequence, $\rho_0, \rho_1, \ldots$, in $\mathbb{N}^\omega$. Given an alphabet Σ, a *Σ-labeled tree* is a pair $\langle T, V \rangle$ where T is a tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. For a Σ-labeled tree $\langle T, V \rangle$ and a path $\rho \subseteq T$, we denote by $V(\rho)$ the sequence $V(\rho_0), V(\rho_1), \ldots$, in Σ^ω. Of special interest to us are Σ-labeled trees in which $\Sigma = 2^{AP}$ for some set AP of atomic propositions. We call such Σ-labeled trees *computation trees*. Note that a computation tree can be viewed as a Kripke structure with infinitely many states. We sometimes refer to satisfaction of temporal logic formulas in a computation tree, meaning their satisfaction in this Kripke structure. Given a set $\mathcal{D} \subset \mathbb{N}$, a $\mathcal{D}$-tree is a computation tree in which all the nodes have degrees in $\mathcal{D}$.

Finite automata on infinite trees (tree automata, for short) run on such Σ-labeled trees. Tree automata were traditionally defined on trees with a fixed branching degree. We generalize here the definition and let tree automata run on trees with varying, yet known in advance, branching degrees. Formally, an automaton on infinite trees is $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, \delta, Q_0, \alpha \rangle$, where Σ, Q, Q^0, and α are as in the case of automata on infinite words, $\mathcal{D} \subset \mathbb{N}$ is a finite set of possible branching degrees, and the transition function δ is $\delta : Q \times \Sigma \times \mathcal{D} \to 2^{Q^*}$, such that for every $q \in Q$, $\sigma \in \Sigma$, and $k \in \mathcal{D}$, we have that $\delta(q, \sigma, k) \subseteq 2^{Q^k}$. Thus, both σ and k are arguments of the transition function. Intuitively, while in word automata δ maps a state and a letter to the

2. *Satisfiability*, where a desired property, specified in temporal logic, is checked to have a model. Formally, the satisfiability problem for linear temporal logics is stated as follows: given a linear temporal logic formula ψ, determine whether there exists a computation π such that $\pi \models \psi$. The satisfiability problem for branching temporal logics is stated as follows: given a branching temporal logic formula φ, determine whether there exists a Kripke structure K such that $K \models \varphi$.

Finding efficient model checking and satisfiability decision procedures has been an active area of research. The following complexity results are known.

	Model checking		Satisfiability	
LTL	PSPACE-complete	[SC85]	PSPACE-complete	[SC85]
CTL	linear time	[CES86]	EXPTIME-complete	[FL79]
CTL*	PSPACE-complete	[EL85a]	2EXPTIME-complete	[ES84, EJ88, VS85]
AFMC	linear time	[Cle93]	EXPTIME-complete	[BB87, FL79]
μ-calculus	NP $\cap$ Co-NP	[EJS93]	EXPTIME-complete	[BB87, FL79]

2.2 Automata on Infinite Objects

A *finite automaton on finite words* is $\mathcal{A} = \langle \Sigma, Q, \delta, Q^0, F \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states. Intuitively, $\delta(q, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Since $\mathcal{A}$ may have several initial states and since the transition function may specify many possible transitions for each state and letter, $\mathcal{A}$ may be *nondeterministic*. If $|Q^0| = 1$ and δ is such that for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\delta(q, \sigma)| = 1$, then $\mathcal{A}$ is a *deterministic* automaton. Given an input word $w = c_0 \cdot c_1 \cdots c_n$ in Σ^*, a *run* of $\mathcal{A}$ on w can be viewed as a function $r : \mathbb{N} \to Q$ where $r(0) \in Q^0$ (i.e., the run starts in one of the initial states) and for every $i \geq 0$, we have $r(i+1) \in \delta(r(i), c_i)$ (i.e., the run obeys the transition function). The run r *accepts* w if $r(n) \in F$. Otherwise, it *rejects*. Note that a nondeterministic automaton can have many runs on w. In contrast, a deterministic automaton has a single run on w. An automaton $\mathcal{A}$ accepts an input word w iff there exists a run r of $\mathcal{A}$ on w such that r accepts w.

When $\mathcal{A}$ runs on an infinite word $w \in \Sigma^\omega$, we can not refer to $r(n)$. Instead, we refer to the set $Inf(r)$ of states that r visits *infinitely often*. Formally,

$$Inf(r) = \{q \in Q \ : \text{ for infinitely many } i \in \mathbb{N}, \text{ we have } r(i) = q\}.$$

As Q is finite, it is guaranteed that $Inf(r) \neq \emptyset$. A *finite automaton on infinite words* (word automaton, for short) is $\mathcal{A} = \langle \Sigma, Q, \delta, Q^0, \alpha \rangle$, where Σ, Q, δ, and Q^0, as well as the definition of

modality. Formally, our translation proceeds as follows. Given a formula ψ, let $\lambda z.g(z)$ be a subformula of ψ in which $g(z)$ includes no fixpoint modalities which are not in a scope of a next modality, and let $g'(z)$ denote $new(g, \lambda, z)(z)$. Note that every ψ that includes some fixpoint modality, includes at least one subformula $\lambda z.g(z)$ as required (e.g., the most internal one). By the definition of the function *new*, all the occurrences of z in $g'(z)$ are in a scope of a next modality. Therefore, all the occurrences of $\lambda z.g'(z)$ in $g'(\lambda z.g'(z))$ are also in a scope of a next modality. By the semantics of μ-calculus, we have that $\lambda z.g(z) = g(\lambda z.g(z))$ for every formula $g(z)$. Thus, as $\lambda z.g(z) = \lambda z.g'(z)$, we can replace $\lambda z.g(z)$ by $g'(\lambda z.g'(z))$ and get an equivalent formula in which all the occurrences of $\lambda z.g'(z)$ are in a scope of a next modality. Doing so, we obtain two things. First, all the occurrences of z in $g'(\lambda z.g'(z))$ are now in a scope of a next modality. Second, all the occurrences of other variables which may have appeared in $\lambda z.g(z)$ not in a scope of a next modality are now in a scope of a next modality. In other words, while the transition from $\lambda z.g(z)$ to $\lambda z.g'(z)$ guarantees that all the occurrences of z are in a scope of a next modality, the transition of $\lambda z.g(z)$ to $g'(\lambda z.g'(z))$ also enables to continue the translation to outer fixpoint modalities. Note that for outermost fixpoint modalities it is suffice to replace $\lambda z.g(z)$ by $\lambda z.g'(z)$.

For example, translation of the formula $\psi = \mu y.(p \vee \nu z.(y \vee (z \wedge AXz)))$ proceeds as follows.

1. $new(y \vee (z \wedge AXz), \nu, z) = y \vee AXz$.
2. Therefore, $\nu z.(y \vee (z \wedge AXz))$ is equivalent to $\nu z.(y \vee AXz)$, which is equivalent to $y \vee AX(\nu z.(y \vee AXz))$.
3. $new(p \vee y \vee AX(\nu z.(y \vee AXz)), \mu, y) = p \vee AX(\nu z.(y \vee AXz))$.
4. Therefore, $\mu y.(p \vee y \vee AX(\nu z.(y \vee AXz)))$ is equivalent to $\mu y.(p \vee AX(\nu z.(y \vee AXz)))$.

□

2.1.3 Fundamental Problems and Known Results

In the context of finite-state programs, temporal logics, both linear and branching, are used in a variety of ways. We mention here two:

1. *Model checking*, where a given finite-state program, modeled by a Kripke structure, is checked with respect to a given property, specified in temporal logic. Formally, the model-checking problem for linear temporal logics is stated as follows: given a Kripke structure $K = \langle AP, W, R, w^0, L \rangle$ and a linear temporal logic formula ψ, determine whether for all paths π in K that start at w^0, we have $\pi \models \psi$. The model-checking problem for branching temporal logics is stated as follows: given a Kripke structure K and a branching temporal logic formula φ, determine whether $K \models \varphi$.

Proof: For technical convenience we assume that the formulas are in positive normal form. We first define a function

$$new : \mu\text{-calculus formulas} \times \{\mu, \nu\} \times APV \rightarrow \mu\text{-calculus formulas}.$$

The formula $new(\varphi, \mu, y)$ is obtained from φ by replacing with ***false*** every occurrence of y that is not in a scope of a modality. Similarly, $new(\varphi, \nu, y)$ is obtained from φ by replacing with ***true*** every occurrence of y that is not in a scope of a modality. Formally, we define $new(\varphi, \lambda, y)$ by induction on the structure of φ as follows:

- $new(y, \mu, y) = \textbf{\textit{false}}$. • $new(y, \nu, y) = \textbf{\textit{true}}$.
- $new(\varphi_1 \wedge \varphi_2, \lambda, y) = new(\varphi_1, \lambda, y) \wedge new(\varphi_2, \lambda, y)$.
- $new(\varphi_1 \vee \varphi_2, \lambda, y) = new(\varphi_1, \lambda, y) \vee new(\varphi_2, \lambda, y)$.
- For all φ which differs from $y, \varphi_1 \wedge \varphi_2$, or $\varphi_1 \vee \varphi_2$, we have $new(\varphi, \lambda, y) = \varphi$.

For example, $new(y \vee p \vee EXy, \mu, y) = p \vee EXy$. We now prove that for all λ, y, and $f(y)$, we have

$$\lambda y.f(y) = \lambda y.new(f, \lambda, y)(y).$$

We consider here the case where $\lambda = \mu$. The proof for the case $\lambda = \nu$ is symmetric. By the semantics of μ-calculus, for every $w \in W$ and valuation $\mathcal{V}$ to the free variables (except y) in $f(y)$, we have that $w \in \mu y.f(y)^K(\mathcal{V})$ if and only if $w \in \bigcap\{W' \subseteq W : f^K(\mathcal{V}[y \leftarrow W']) \subseteq W'\}$. For every $W' \subseteq W$, we prove that

$$f^K(\mathcal{V}[y \leftarrow W']) \subseteq W' \text{ if and only if } new(f, \mu, y)^K(\mathcal{V}[y \leftarrow W']) \subseteq W'.$$

Assume first that $f^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$. As for every valuation $\mathcal{V}'$ we have $new(f, \mu, y)^K(\mathcal{V}') \subseteq f^K(\mathcal{V}')$, clearly $new(f, \mu, y)^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$. For the second direction, we assume that f is given in a disjunctive normal form. We prove that if $new(f, \mu, y)^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$, then for every w, if there exists a conjunction g in f such that $w \in g^K(\mathcal{V}[y \leftarrow W'])$, then $w \in W'$. Let g be such that $w \in g^K(\mathcal{V}[y \leftarrow W'])$. We consider two cases:

1. If $g = new(g, \mu, y)$, then, as $new(f, \mu, y)^K(\mathcal{V} \cup \{\langle y, W' \rangle\}) \subseteq W'$, we are done.
2. Otherwise, there is a conjunct y in g and it must be that $g^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$. Hence, we are also done.

The definition of *new* guarantees that all the occurrences of y in $new(f, \lambda, y)$ are in a scope of *some* modality. We, however, want all the occurrences of y to be in a scope of a *next* modality. As a solution, we will make sure that each fixpoint modality is in a scope of a next

For example, for $\varphi = \mu y.(q \vee (p \wedge EXy))$, $cl(\varphi) = \{\varphi, q \vee (p \wedge EX\varphi), q, p \wedge EX\varphi, p, EX\varphi\}$. As proved in [Koz83], for every μ-calculus formula φ, the size of $cl(\varphi)$ is linear in $|\varphi|$.

Given a Kripke structure $K = \langle AP, W, R, w^0, L\rangle$, and a set $\{y_1, \ldots, y_n\}$ of free variables, a *valuation* $\mathcal{V} = \{\langle y_1, W_1\rangle, \ldots, \langle y_n, W_n\rangle\}$ is an assignment of subsets of W to the variables $\{y_1, \ldots, y_n\}$. For a valuation $\mathcal{V}$, a variable y, and a set $W' \subseteq W$, we denote by $\mathcal{V}[y \leftarrow W']$ the valuation obtained from $\mathcal{V}$ by assigning W' to y. A formula φ with free variables $\{y_1, \ldots, y_n\}$ is interpreted as a mapping φ^K from valuations to 2^W. Thus, $\varphi^K(\mathcal{V})$ denotes the set of states that satisfy φ with the valuation $\mathcal{V}$. The mapping φ^K is defined inductively as follows:

- $\boldsymbol{true}^K(\mathcal{V}) = W$ and $\boldsymbol{false}^K(\mathcal{V}) = \emptyset$.
- For an atomic proposition $p \in AP$, we have $p^K(\mathcal{V}) = \{w \in W : p \in L(w)\}$.
- For an atomic proposition variable $y_i \in APV$, we have $y_i^K(\mathcal{V}) = W_i$.
- $(\neg\varphi_1)^K(\mathcal{V}) = W \setminus \varphi_1^K(\mathcal{V})$.
- $(\varphi_1 \vee \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cup \varphi_2^K(\mathcal{V})$.
- $(EX\varphi_1)^K(\mathcal{V}) = \{w \in W : \exists w' \in \varphi_1^K(\mathcal{V}) \text{ and } \langle w, w'\rangle \in R\}$.
- $(\mu y.f(y))^K(\mathcal{V}) = \bigcap\{W' \subseteq W : f^K(\mathcal{V}[y \rightarrow W']) \subseteq W'\}$.

Note that no valuation is required for a sentence. For a state $w \in W$ and a sentence φ, we say that $w \models \varphi$ iff $w \in \varphi^K$. For example, the μ-calculus formula $\mu y.(q \vee (p \wedge EXy))$ is equivalent to the CTL formula $EpUq$.

A μ-calculus formula is *alternation free* if there are no occurrences of ν (μ) on any syntactic path from an occurrence of μy (νy) to an occurrence of y. For example, the formula $\mu x.(p \vee \mu y.(x \vee EXy))$ is alternation free and the formula $\nu x.\mu y.((p \wedge x) \vee EXy)$ is not alternation free. The *alternation-free μ-calculus* (AFMC) is a subset of μ-calculus containing only alternation-free formulas.

A μ-calculus formula is *guarded* if for all $y \in APV$, all the occurrences of y that are in a scope of a fixpoint modality λ are also in a scope of a next modality (which is in the scope of λ too). Thus, a μ-calculus sentence is guarded if for all $y \in APV$, all the occurrences of y are in a scope of a next modality. For example, the formula $\mu y.(p \vee EXy)$ is guarded and the formula $\mu y.(p \vee y)$ is not guarded. We assume that all μ-calculus formulas are guarded. By the theorem below, stated without a proof in [BB87], this can be done without loss of generality.

Theorem 2.1 *Given a μ-calculus formula, we can construct, in linear time, an equivalent guarded formula.*

($L_1 \geq L_2$), provided that for every formula φ_2 of L_2, there exists an equivalent formula φ_1 of L_1. Also, L_1 is *as expressive as* L_2 ($L_1 = L_2$), if both $L_1 \geq L_2$ and $L_2 \geq L_1$, L_1 is *strictly more expressive than* L_2 ($L_1 > L_2$), if both $L_1 \geq L_2$ and $L_2 \not\geq L_1$, and L_1 and L_2 are *incomparable* ($L_1 \neq L_2$) if both $L_1 \not\geq L_2$ and $L_2 \not\geq L_1$.

2.1.2 The Propositional μ-calculus

The *propositional μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators. Specifically, we consider a μ-calculus where formulas are constructed from Boolean propositions with Boolean connectives, the temporal operator EX, as well as least (μ) fixpoint operators. Formally, given a set AP of atomic proposition constants and a set APV of atomic proposition variables, a μ-calculus formula is either:

- ***true***, ***false***, or p for all $p \in AP$.
- y, for all $y \in APV$.
- $\neg\varphi_1$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are μ-calculus formulas.
- $EX\varphi_1$, where φ_1 is a μ-calculus formula.
- $\mu y.f(y)$, where $f(y)$ is a μ-calculus formula in which all the free occurrences of y fall under an even number of negation.

We use $\nu y.f(y)$ to abbreviate $\neg\mu y.\neg f(y)$ and we say that an atomic proposition variable y is *free* if it is not in a scope of μy. A *sentence* is a formula that contains no free atomic proposition variables. We call AX and EX *next modalities*, and we call μ and ν *fixpoint modalities*. We say that a μ-calculus formula is a *μ-formula* (*ν-formula*), if it is of the form $\mu y.f(y)$ ($\nu y.f(y)$). We use λ to denote a fixpoint modality μ or ν. For a λ-formula $\lambda y.f(y)$, the formula $f(\lambda y.f(y))$ is obtained from $f(y)$ by replacing each occurrence of y with $\lambda y.f(y)$.

The closure $cl(\varphi)$ of a μ-calculus sentence φ is the smallest set of μ-calculus sentences that satisfies the following:

- $\varphi \in cl(\varphi)$.
- If $\varphi_1 \wedge \varphi_2 \in cl(\varphi)$, then $\varphi_1 \in cl(\varphi)$ and $\varphi_2 \in cl(\varphi)$.
- If $\neg\varphi_1 \in cl(\varphi)$ or $EX\varphi_1 \in cl(\varphi)$, then $\varphi_1 \in cl(\varphi)$.
- If $\mu y.f(y) \in cl(\varphi)$, then $f(\mu y.f(y)) \in cl(\varphi)$.

relation that must be total in its first element (i.e., for each $w \in W$ there exists at least one w' with $\langle w, w' \rangle \in R$), w^0 is an initial state, and $L : W \to 2^{AP}$ maps each state to a set of atomic propositions true in this state. For $\langle w_1, w_2 \rangle \in R$, we say that w_2 is a *successor* of w_1 and w_1 is a *predecessor* of w_2. A *path* in K is an infinite sequence of states $\pi = w_0, w_1, \ldots$, such that for all $i \geq 0$, we have $\langle w_i, w_{i+1} \rangle \in R$. We denote by $bd(w)$ the branching degree of w; i.e., the number of successors that w has.

We use $w \models \varphi$ to indicate that a state formula φ holds at state w (assuming an agreed Kripke structure K). The relation $\models$ is inductively defined as follows (the relation $\pi \models \psi$ for a path formula ψ is the same as for ψ in LTL, adding the case where ψ is a state formula):

- For all w, we have that $w \models \textbf{\textit{true}}$ and $w \not\models \textbf{\textit{false}}$.
- For an atomic proposition $p \in AP$, $w \models p$ iff $p \in L(w)$.
- $w \models \neg\varphi_1$ iff $w \not\models \varphi_1$.
- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$.
- $w \models E\psi_1$ iff there exists a path $\pi = w_0, w_1, \ldots$ such that $w_0 = w$ and $\pi \models \psi_1$.
- $\pi \models \varphi$ for a state formula φ iff $w_0 \models \varphi$.

For a Kripke structure K, and a CTL* formula φ, we say that $K \models \varphi$ iff $w^0 \models \varphi$.

The logic *CTL* is a restricted subset of CTL*. In CTL, the temporal operators X, U, and their negations must be immediately preceded by a path quantifier. Formally, it is the subset of CTL* obtained by restricting the path formulas to be $X\varphi_1$, $\varphi_1 U \varphi_2$, or their negations, where φ_1 and φ_2 are CTL state formulas.

We say that a CTL formula φ is an *U-formula* if it is of the form $A\varphi_1 U \varphi_2$ or $E\varphi_1 U \varphi_2$. The subformula φ_2 is then called the *eventuality* of φ. Similarly, φ is a *$\tilde{U}$-formula* if it is of the form $A\varphi_1 \tilde{U} \varphi_2$ or $E\varphi_1 \tilde{U} \varphi_2$.

A *positive normal form* of temporal logic formulas is a form in which negations are applied only to atomic propositions. A positive normal form can be obtained by pushing negations inward as far as possible using De Morgan's laws and dualities. Translating a formula to a positive normal form at most doubles the length of the formula. For technical convenience, we often assume that formulas are given in a positive normal form.

We say that two formulas φ_1 and φ_2 are *equivalent* ($\varphi_1 \sim \varphi_2$) if for every Kripke structure K, we have $K \models \varphi_1$ iff $K \models \varphi_2$. We say that two path formulas ψ_1 and ψ_2 are *congruent* ($\psi_1 \approx \psi_2$) if for every Kripke structure K and path π in it, we have $\pi \models \psi_1$ iff $\pi \models \psi_2$. When comparing expressive power of two logics L_1 and L_2, we say that L_1 is *more expressive than* L_2

- $\pi \models X\psi_1$ iff $\pi^1 \models \psi_1$.
- $\pi \models \psi_1 U \psi_2$ iff there exists $k \geq 0$ such that $\pi^k \models \psi_2$ and $\pi^i \models \psi_1$ for all $0 \leq i < k$.

The logic *CTL** is a branching temporal logic. A path quantifier, E ("for some path") can prefix an assertion composed of an arbitrary combination of linear time operators. There are two types of formulas in CTL*: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*, whose satisfaction is related to a specific path. Formally, let AP be a set of atomic proposition names. A CTL* state formula is either:

- ***true***, ***false***, or p, for $p \in AP$.
- $\neg\varphi_1$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are CTL* state formulas.
- $E\psi_1$, where ψ_1 is a CTL* path formula.

A CTL* path formula is either:

- A CTL* state formula.
- $\neg\psi_1$, $\psi_1 \vee \psi_2$, $X\psi_1$, or $\psi_1 U \psi_2$, where ψ_1 and ψ_2 are CTL* path formulas.

The logic CTL* consists of the set of state formulas generated by the above rules. We denote the size of a formula φ by $|\varphi|$ and we use the following abbreviations in writing formulas:

- $\wedge, \rightarrow$, and $\leftrightarrow$, interpreted in the usual way.
- $A\psi = \neg E \neg \psi$ ("in all paths").
- $F\psi = \textbf{\textit{true}} U \psi$ ("eventually", and the "F" comes from "Future").
- $G\psi = \neg F \neg \psi$ ("always", and the "G" comes from "Globally").
- $\psi_1 \tilde{U} \psi_2 = \neg((\neg\psi_1) U (\neg\psi_2))$ (duality for the Until operator).

The *closure* $cl(\psi)$ of a CTL* formula ψ is the set of all state subformulas of ψ (including ψ but excluding ***true*** and ***false***). For example, $cl(E(pU(AXq))) = \{E(pU(AXq)), p, AXq, q\}$. It is easy to see that the size of $cl(\psi)$ is linear in the size of ψ. Note that since CTL* formulas are state formulas, we could write the formula $E(pU(AXq))$ also as $EpU(AXq)$, causing no ambiguity in its parsing. For technical convenience, we will omit such unnecessary brackets in the sequel.

We define the semantics of CTL* with respect to a *Kripke structure* $K = \langle AP, W, R, w^0, L\rangle$, where AP is the set of atomic propositions, W is a set of states, $R \subseteq W \times W$ is a transition

Chapter 2

Basic Definitions

2.1 Temporal Logics

State-exploration methods are applicable with respect to finite-state system. Since each state in such a system is characterized by a finite amount of information, this information can be described by certain *atomic propositions*. In this section we define *propositional* temporal logics, where formulas are defined over a set AP of atomic propositions.

2.1.1 The Temporal Logics LTL, CTL*, and CTL

The logic *LTL* is a linear temporal logic. Formulas of LTL are constructed from a set AP of atomic proposition using the usual Boolean operators and the temporal operators X ("next time") and U ("until"). Formally, given a set AP, an LTL formula is defined as follows:

- ***true***, ***false***, or p, for $p \in AP$.
- $\neg\psi_1$, $\psi_1 \vee \psi_2$, $X\psi_1$, or $\psi_1 U \psi_2$, where ψ_1 and ψ_2 are LTL formulas.

We define the semantics of LTL with respect to a *computation* $\pi = \sigma_0, \sigma_1, \sigma_2, \ldots$, where for every $j \geq 0$, σ_j is a subset of AP, denoting the set of atomic propositions that hold in the j's position of π. We denote the suffix $\sigma_j, \sigma_{j+1}, \ldots$ of π by π^j. We use $\pi \models \psi$ to indicate that an LTL formula ψ holds in the path π. The relation $\models$ is inductively defined as follows:

- For all π, we have that $\pi \models$ ***true*** and $\pi \not\models$ ***false***.
- For an atomic proposition $p \in AP$, $\pi \models p$ iff $p \in \sigma_0$.
- $\pi \models \neg\psi_1$ iff $\pi \not\models \psi_1$.
- $\pi \models \psi_1 \vee \psi_2$ iff $\pi \models \psi_1$ or $\pi \models \psi_2$.

power and ease of use within the complexity limits enforced by the user. In the second part of this work, we consider three different ways of augmenting branching temporal logics. Each of the three ways makes branching temporal logic a more useful specification language.

We start by showing that model-checking tools which handle the branching temporal logic CTL can be very easily extended to handle a more expressive and a much more convenient logic, with hardly any change in their complexity. We then turn to consider the augmentation of branching temporal logics with *existential quantification over atomic propositions*. Such an augmentation enables the user to refer, in the specification, to atoms that are not part of the system and that are added by him; e.g., in order to mark all the even positions of the system. For linear temporal logics, quantification over atomic propositions is considered in [Sis83] and [Wol83] and is suggested as a useful extension for these logics. For branching temporal logics, we show that existential quantification over atomic propositions is sufficient to make the logics very expressive yet it dramatically increases their model-checking complexity. Quantification over atomic propositions has a lot in common with temporal modalities that refer to the *past*. Augmenting linear temporal logics with such modalities extends neither their expressive power [GPSS80] nor the complexity of their model-checking problem [LPZ85, Var88]. For branching temporal logics, we show that the picture is diversified. We formalize the connection between quantification over atomic propositions and temporal modalities that refer to the past and study each of them in terms of its practicality and complexity.

Motivated by decision problems in mathematics and logic, Büchi, McNaughton, and Rabin developed a framework for reasoning about infinite words and infinite trees [Büc62, McN66, Rab69]. Beyond being beautiful, the framework proved to be very powerful. Automata, and their tight relation to second order monadic logics were the key to the solution of several fundamental decision problems in mathematics and logics [Tho90].

For linear temporal logics, a close and fruitful relationship with the theory of automata on infinite words has been developed [VW86a, VW94]. The basic idea of Vardi and Wolper is to associate with each formula an automaton on infinite words that accepts exactly all the computations that meet the behavior specified by the formula. This enables the reduction of linear temporal logic problems, such as satisfiability and model checking, to known automata-theoretic problems, yielding clean and optimal algorithms. Furthermore, these reductions are very helpful for implementing state-exploration methods, and are paving the way to techniques such as *on-the-fly* verification [VW86a, JJ89, CVWY92] that help coping with the state-explosion problem.

For branching temporal logics, the automata-theoretic counterpart are automata on infinite trees. By reducing the satisfiability problem to the nonemptiness problem for these automata, optimal decision procedures have been obtained for various branching temporal logics [ES84, SE84, Eme85, VW86b, EJ88]. Nevertheless, the automata-theoretic approach has long been thought to be inapplicable for branching-time model checking. The reason, essentially, lies in the gap between the complexities of the satisfiability problem and the model-checking problem for branching temporal logics. Unlike linear temporal logics, where the complexities of these problems coincide, the model-checking problem for a branching temporal logic is typically much easier than its satisfiability problem. It seemed that automata can not compete with this gap and hence, they are practically useless for model checking. This problem stood open since 1986.

In this work we solve this problem, using *alternating tree automata*. From a theoretical point of view, solving this problem completes the picture of using automata for both satisfiability and model checking of both linear and branching temporal logics. From a practical point of view, solving this problem enables the improvement of the *space complexity* of the model-checking problem. Evidently, such an improvement is an important step in the struggle with the state-explosion problem. Moreover, it turned out that automata can be used to improve the complexity also of *fair* model checking, where a system is checked to be correct under certain fairness constraints, and of *modular* model checking, where a system is checked to be correct by checking its components.

Improving branching-time model-checking complexity is only one way to make this method practical. Naturally, there is a trade-off between the *expressive power* of a temporal logic and the *complexity* of its model-checking problem: the more a temporal logic is expressive, the higher its model-checking complexity is. A wise model checker should enable the maximal expressive

assertions. Interestingly, for some logics, the truth of an assertion can be more involved than a simple Boolean value. Beyond being true or false, an assertion can be, say, "possibly true" (i.e., true in some other world) or "known as true" (i.e., true in this world according to the information of some other world). Philosophers and logicians knew to distinguish between these levels of truth and developed *modal logics* which study them [HC68]. What they did not know, however, is that some hundreds years later, in 1977, Pnueli will suggest their modal logics as a formal language for specifying and verifying correctness of computerized systems [Pnu77].

The special type of modal logic which was advocated as appropriate for this task is *temporal logic*. In a temporal logic, we augment a conventional logic with *temporal modalities*, making it possible to describe an occurrence of events in time. For example, using the temporal modalities *always* and *eventually*, we can specify the behavior "if x holds in all future moments then there is a future moment in which y holds too". Thus, as opposed to traditional formalisms where the specification can only relate the initial state and the final state of a system [Flo67, Hoa69], temporal logic is well-suited to describe the on-going behavior of *nonterminating* and *interactive* systems.

Two possible views regarding the nature of time induce two types of temporal logics [Lam80]. In *linear* temporal logics, time is treated as if each moment in time has a unique possible future. Thus, linear temporal logic formulas are interpreted over linear sequences and we regard them as describing a behavior of a single computation of a system. In *branching* temporal logics, each moment in time may split into various possible futures. Accordingly, the structures over which branching temporal logic formulas are interpreted can be viewed as infinite trees, each describing the behavior of the possible computations of a nondeterministic system.

In this work we study state-exploration methods that use branching temporal logics as their formal specification language. In this paradigm, which we call *branching-time model checking*, formal verification means determining whether a structure satisfies a branching temporal logic formula. Model-checking techniques were introduced in the early 80's by Clarke and Emerson [CE81], and Quielle and Sifakis [QS81]. Since then, branching temporal logics have acquired a considerable reputation for having a low model-checking complexity. The last decade was a blooming period to many theoretical research in this area [Eme90], yielding branching-time model-checking tools that can handle systems with more than 10^{120} states [Bro86, McM93, CGL93]. The last decade was less blooming for many bugs [BCD85, BCDM86, DC86, CGH+95], being traced by these tools. Slowly, yet persistently, industrial companies are becoming aware of the importance and the potentiality of formal verification. Branching-time model-checking tools are incorporated into industrial development of new designs [BBG+94], forming an active and exciting area of research, where theory and practice go together, enjoying each other.

This work employs the theory of *automata on infinite objects* for the benefit of branching-time model checking. Finite automata on infinite objects were first introduced in the 60's.

development of complex computerized systems requires more reliable methods: it is very likely that a complex system will contain bugs and it is also likely that tests of a complex system will not trace all its bugs. Such a more reliable method is *formal verification*.

In formal verification, we verify that a system meets a desired behavior by checking that a mathematical model of the system satisfies a formal specification that describes the behavior. Formal verification has two advantages over testing. First, formal verification is *exhaustive*. As such, it checks the system with respect to all its possible behaviors in any possible environment. Second, using a *formal specification language*, there is no ambiguity in what it means for a behavior of the system to be correct. A behavior is correct precisely when it satisfies the formal specification. In addition, formal verification can be performed on an abstract model of the system or on separate components of it. Hence, it can be an integral part of the development process, tracing bugs when they are still harmless and restrained.

Formal verification consists of two main verification streams. One accommodates the more veteran *proof-based* methods and the other accommodates the relatively new *state-exploration* methods. In proof-based methods, we attribute the model of the system with assertions in a formal specification language and construct a proof that relates these assertions [Fra92]. Since the specification language can be very expressive (e.g., when it subsumes the full power of mathematics), proof-based methods are very strong, in the sense that they are applicable to a wide range of systems and behaviors. Many users, however, find proof-based methods unattractive. Indeed, it is hard to get attracted by a method which requires supply and proof of intricate assertions. The development of *theorem provers* and *proof checkers* eases this difficulty by making the verification task an interactive process in which the user is as passive as possible [BM83, Gor85, SOR93, MAB+94]. Unfortunately, nowadays users are are still quite active. At least, they are sufficiently active not to make proof-based methods widely used.

In state-exploration methods we navigate through the model of the system and sniff around. Therefore, state-exploration methods are restricted to *finite-state* systems (i.e., systems in which the variables range over finite domains). Circuits and a large number of communication and synchronization protocols are in essence finite-state systems. The exploratory nature of state-exploration methods makes them fully automatic, convenient to use, and very attractive. Furthermore, when the model of the system does not satisfy its required specification, state-exploration methods can often produce a buggy script of the system execution, helping the user to trace the bug elegantly. Unfortunately, the exploratory nature of state-exploration methods also makes them very sensitive to the size of the model. Since modeling a system with several components involves an exponential blow-up in the number of the components, this sensitivity is really a painful problem. So painful, that the so-called *state explosion* problem is one of the most challenging issues these days in the area of computer-aided verification.

Formal verification is strongly related to the theory of *logic*. Logic studies the truth of

Chapter 1

Introduction

Consider the following procedure which gets as input two variables x and y and calculates their minimum in the output variable min:

procedure $minimum(x, y; min)$;
  if $(x \neq 97)$ and $(x \leq y)$ **then** $min := x$
  else $min := y$

Some readers may suspect that even though the procedure is called *minimum*, it does not really calculate the minimum of x and y. To increase these readers' belief in the correctness of the procedure, we can show them that it works correctly with $\langle x, y \rangle = \langle 3, 7 \rangle$. We can even suggest them to *test* the procedure with respect to random inputs. If we are lucky, all their random input pairs $\langle x, y \rangle$ will be such that $x \neq 97$ or $y \leq x$. Naturally, each such test will increase the readers' belief in the correctness of the procedure and, after several successful tests, most readers will get convinced that the procedure is correct.

Of course, an intelligent reader could have come with a counter-example to the correctness of the procedure[1], but as long as we assume that our readers are fully automatic (as computers that test procedures are), the best they can do is to test the procedure again and again. Doing so, they either come across an unexpected behavior of the procedure and conclude that it is buggy or, they decide, after sufficiently (according to their standards) many tests that the procedure is correct. This is a very unpleasant reality. To see how unpleasant it is one only has to change the procedure name from *minimum* to, say, *max_elevator_weight*. Indeed, our world is full of merciless bugs. While testing has once been considered a satisfying method to trace them and to get rid of them while they are still small and under control, today's rapid

[1] Say, $\langle x, y \rangle = \langle 97, 99 \rangle$, for our less intelligent readers.

modalities that refer to the past. Each of the three ways makes branching temporal logic a more convenient specification language. We examine their both theoretical and practical aspects.

Abstract

The thesis considers *formal verification* of computerized systems. In formal verification, we verify that a system meets a desired behavior by checking that a mathematical model of the system satisfies a formal specification that describes the behavior. The special method of formal verification that we consider is *branching-time model checking*. In this method, formal verification means determining whether a Kripke structure satisfies a branching temporal logic formula.

Translating linear temporal logic formulas to automata on infinite words has proven to be an effective approach for implementing linear-time model-checking and for obtaining many extensions and improvements to this verification method. On the other hand, for branching temporal logics, automata-theoretic techniques have long been thought to introduce an exponential penalty, making them essentially useless for model-checking. Efforts to employ the theory of automata on infinite trees for the benefit of branching-time model checking started at the 80's, but with no success. In this thesis we show that *alternating tree automata* are the key to a comprehensive automata-theoretic framework for branching temporal logics.

Using the automata-theoretic framework we are able to improved the space complexity of branching-time model-checking. In particular, we show that alternating tree automata provide a PSPACE procedure for CTL* model checking of *concurrent programs*, and provide an explanation why this bound can not be achieved for μ-calculus model checking and even for its alternation-free fragment. We show that the efficient bounds hold also for *fair* model-checking, where a system is checked to be correct under certain fairness constraints, and for *modular* model checking, where a system is checked to be correct by checking its components.

Improving branching-time model-checking complexity is only one way to make this method practical. Naturally, there is a trade-off between the *expressive power* of a temporal logic and the *complexity* of its model-checking problem: the more a logic is expressive, the higher its model-checking complexity is. A wise model checker should enable the maximal expressive power and ease of use within the complexity limits enforced by the user. In the second part of the thesis, we consider three different ways of extending conventional branching temporal logics: more liberal syntax, existential quantification over atomic propositions, and temporal

Contents

Acknowledgments

I find the writing of this acknowledgment very difficult. First, I know that the acknowledgment is the only part of the thesis that most people are going to read. Second, there is a tendency, when writing these things, to thank everybody, and everything ("...and my dog, Muki, whose chewing up of countless drafts forced me to ever higher levels of performance..."), and to do it in a very serious and mature style ("...Muki, who has shaped my growth as a scientist and as a person..."), which I don't have. Third, and most important, I believe that those whom I want to thank know exactly how deep my thanks are, with or without an acknowledgment. Therefore, the only goal of this acknowledgment is to describe the reasons for these thanks.

As an undergraduate student, I attended four courses by Orna Grumberg. I reached formal verification since, and only since, I wanted her to be my supervisor. I had no doubts that it would be a successful bet, but I didn't estimate that it would be so successful and so significant. In my first year as a graduate student I had daily meetings with Orna. Meetings which could easily last all day long. I spent so many (marvelous) hours in Orna's office that I know, and I will miss, every spot there. The knowledge, interest, trust, and freedom that Orna gave me all along the last four years are invaluable. Indeed, I was very fortunate.

I have no idea how this thesis and its author would have looked like unless I met Moshe Vardi. The more I am thinking about it, the more I realize how different (and less satisfying) their fate could have been. Awarding me with more than 1000 (thousand!) mails, Moshe made the last three years extremely exciting and fruitful. Logins in the morning are much more animating when I know that mails from Moshe are probably waiting. And they waited, regularly, professional, spurring, and lovely, and became a part of my life (and of this thesis). Together with the mails, Moshe stuffed me with big envelopes full of papers and, during our meetings, with an inspiration to read them. No wonder then, that each of my meetings with Moshe turned to be a happy landmark in my research.

And here comes Pierre Wolper. Pierre's clear and precise style, his enthusiasm to share his vast knowledge, and his patient to my narrow knowledge, they all make a collaboration with Pierre an enlightening experience. Keeping in mind that it was [VW86a] that introduced me to the enchanting world of automata, it is just natural that an automata-theoretic collaboration with Vardi and Wolper was so fascinating.

To make my fortune utter, I had the pleasure to meet the sage. Working with Amir Pnueli is so blissful and instructive that it is worth, far and away, his weakness: I nagged almost always, and Amir gave me infinitely many answers, solutions, opinions, techniques, ideas, and other good reasons to nag again and again. This is, definitely, more than fair.

Finally, it was great to share an office with Amos Beimel (and his helpful knowledge in complexity theory), to meet and work with Muli Safra and Tom Henzinger, to get from Allen Emerson quick replies to all my queries, and to intensively consult, communicate, and smile with David Harel.

This research was carried out in the Faculty of Computer Science
under the supervision of Dr. Orna Grumberg

Model Checking for Branching-Time Temporal Logics

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Science

Orna Kupferman

Submitted to the Senate of the Technion – Israel Institute of Technology
Sivan 5754 Haifa June 1995