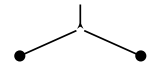
Model Checking Finite Paths and Trees

Dissertation zur Erlangung des Grades des Doktors der Naturwissenschaften der Naturwissenschaftlich-Technischen Fakultäten der Universität des Saarlandes

Lars Kuhtz

Saarbrücken, 2010



Abstract

This thesis presents efficient parallel algorithms for checking temporal logic formulas over finite paths and trees. We show that LTL path checking is in $\mathsf{AC}^1(\mathsf{logDCFL})$ and CTL tree checking is in $\mathsf{AC}^2(\mathsf{logDCFL})$. For LTL with past-time and bounded modalities, which is an exponentially more succinct logic, we show that the path checking problem remains in $\mathsf{AC}^1(\mathsf{logDCFL})$. Our results provide a foundation for efficient algorithms of various applications in monitoring, testing, and verification as well as for query processing for tree-datastructures, e.g. XML documents.

The presented path and tree checking algorithms are based on efficient parallel evaluation strategies for monotone Boolean circuits. We reduce the evaluation of product circuits to the problem of evaluating one-input-face monotone planar Boolean circuits: for a monotone Boolean circuit that is a product of a tree and a path, we provide an AC^1-reduction; for a monotone Boolean circuit that is a product of two trees, we provide an AC^2-reduction.

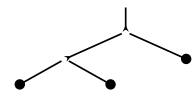
We develop a classification of Kripke structures with respect to the complexity of LTL model checking: Kripke structures for which the problem is PSPACE-complete, Kripke structures for which the problem is coNP-complete, and Kripke structures for which the problem is in NC.

Zusammenfassung

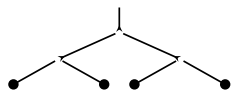
Wir präsentieren effiziente parallele Algorithmen zum Überprüfen der Erfülltheit von temporal logischen Formeln auf Pfaden und Bäumen. Wir zeigen, dass für die Logik LTL das Überprüfen von Ausführungspfaden in der Komplexitätsklasse $\mathsf{AC}^1(\mathsf{logDCFL})$ liegt. Für die Logik CTL ist das Überprüfen von Bäumen in $\mathsf{AC}^2(\mathsf{logDCFL})$. Für Erweiterungen von LTL mit Vergangenheit und beschränkten zeitlichen Modalitäten beweisen wir, dass Pfade ebenfalls in $\mathsf{AC}^1(\mathsf{logDCFL})$ überprüft werden können, obwohl die Logik exponentiell kompakter ist als einfaches LTL. Unsere Resultate bielden eine Grundlage für effiziente Algorithmen für verschiedene Anwendungen in den Bereichen der Systemüberwachung, des Testens und der Verifikation sowie für die Anfragebearbeitung für Baumdatenstrukturen, wie zum Beispiel XML Dokumente.

Die präsentierten Algorithmen zum Überprüfen von Pfaden und Bäumen basieren auf effizient parallelen Strategien zur Evaluierung von monotonen Boolschen Schaltkreisen. Wir reduzieren die Evaluierung von Produkt-Schaltkreisen auf das Problem der Evaluierung von monoton planaren Boolschen Schaltkreisen, bei denen sich alle Eingaben auf dem äußeren Rand befinden. Für monotone Boolsche Schaltkreise, die das Produkt von einem Baum und einem Pfad sind, geben wir eine AC^1-Reduktion an. Für monotone Boolsche Schaltkreise, die das Produkt von zwei Bäumen sind, geben wir eine AC^2-Reduktion an.

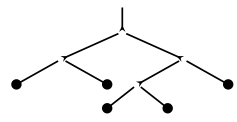
Wir entwickeln eine Klassifizierung von Kripkestrukturen im Hinblick auf die Komplexität des Erfülltheitsproblems für LTL: Kripkestrukturen, für die das Problem PSPACE-vollständig ist, Kripkestrukturen, für die das Problem coNP-vollständig ist, und Kripkestrukturen, für die das Problem in NC liegt.



Contents



List of Figures



Chapter 1

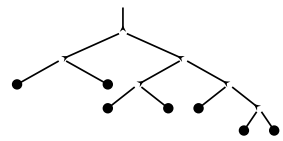
Introduction

The past decades have brought significant advances in the computer-aided verification of computer systems. Many hardware systems and communication protocols can be modelled as small finite-state structures, which can be analyzed automatically. The fact, however, that the state space of a complex system grows exponentially with the number of its components, represents a complexity-theoretic barrier for all algorithmic methods that attempt to analyze the complete set of all possible system behaviors. This so-called state-space explosion problem has motivated a great variety of approaches to simplify the problem, for example using heuristics, abstraction, and compositional verification. The single most successful approach, applied in areas such as testing, runtime verification, and Monte-Carlo verification, has, however, been to limit the attention from a set of hypothetical behaviors to one particular behavior, as observed for example during an execution of the system.

The topic of this thesis is the complexity-theoretic and algorithmic benefits that result from this reduction. We investigate both the *linear-time* setting, where we consider paths, i.e., linear sequences of states, and the *branching-time* setting, where we consider a tree of states called the computation tree.

The problems of checking paths and trees are among the few fundamental model checking problems whose complexity is still open [71]: on the one hand, the standard automata-based algorithms run in polynomial time or worse [28, 71, 41, 18]; on the other hand, the only known lower bound is NC^1 [23], the complexity of evaluating Boolean expressions.

In this thesis, we break the barrier from inherently sequential to efficiently parallelizable algorithms. We improve the upper bound on the complexity of checking linear-time temporal logic (LTL) over paths from P to AC^2, and the complexity of of computation tree logic (CTL) over trees from P to SAC^3. In



order to obtain these results we depart from the classic automata-based setting and instead study model checking in the setting of circuit evaluation problems.

1.1 Model Checking Finite Paths and Trees

LTL path checking. Linear-time temporal logic (LTL) is the standard specification language to describe properties of reactive computation paths. The problem of checking whether a given finite path satisfies an LTL formula plays a key role in monitoring and runtime verification [41, 28, 20, 5, 15], where individual paths are checked either online, during the execution of the system, or offline, for example based on an error report. Similarly, path checking occurs in testing [6] and in several static verification techniques, notably in Monte-Carlo-based probabilistic verification, where large numbers of randomly generated sample paths are analyzed [92].

Somewhat surprisingly, given the widespread use of LTL, the complexity of the path checking problem is still open [71]. The established upper bound is P: The algorithms in the literature traverse the path sequentially (cf. [28, 71, 41]); by going backwards from the end of the path, one can ensure that, in each step, the value of each subformula is updated in constant time, which results in bilinear running time. The only known lower bound is NC^1 [23], the complexity of evaluating Boolean expressions [16]. The large gap between the bounds is especially unsatisfying in light of the recent trend to implement path checking algorithms in hardware, which is inherently parallel. For example, the IEEE standard *Property Specification Language (PSL)* [45], that subsumes LTL has become part of the hardware description language VHDL, and several tools [20, 15] are available to synthesize hardware-based monitors from assertions written in PSL. Can we improve over the sequential approach by evaluating entire blocks of path positions in parallel? In the thesis we show that LTL path checking can indeed be parallelized efficiently[1].

Modern specification languages like PSL include concepts that stem from different mathematical frameworks. Besides the elements of a classical hardware description languages it includes concepts borrowed from formal languages, namely

[1] We say a problem can be decided *efficiently in parallel* if it is in NC, i.e. if it can be decided by a uniform family of polynomial size Boolean circuits of poly-logarithmic depth [32]. The term is sometimes used with the stronger meaning that the total amount of work (the number of gates in a circuit) is linear in the time complexity (the number of computation steps) of the best sequential algorithm. In contrast, following [32] we allow a polynomial blow-up.

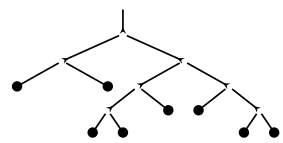
There are also different intuitive characterizations of the class NC. Papadimitrou e.g. is less enthusiastic about the relevance of NC by calling it the "problems satisfactorily solved by parallel computers" [75]. Another common description for NC is to call it the class of problems with "highly parallel algorithms" [39].

extended regular expressions and concepts from temporal logics, namely LTL and CTL along with different extensions of LTL.

From an expressiveness point of view, LTL–in the field of temporal logics– corresponds to regular expressions –in the field of formal languages. They both describe the regular languages or a fragment thereof. What are the trade-offs between these formalisms? How do they compare from a complexity point of view? The path checking problem for temporal logics corresponds to the membership problem for formal languages. For regular expressions and their most prominent derivatives the complexity of the membership problem is well-understood: it is in nondeterministic logspace (NL) for (normal) regular expressions [48], it is in $\mathsf{logCFL} \subseteq \mathsf{NC}$ for semi-extended regular expressions [77], and it is complete for polynomial time (P) for star-free regular expressions and semi-extended regular expressions [76]. Of particular interest is the comparison of LTL to the star-free regular expressions, since they have the same expressive power as LTL on finite paths [66]. With $\mathsf{AC}^1(\mathsf{logDCFL})$ vs. P, our result demonstrates a computational advantage for LTL. In particular it is interesting that all classes of regular expressions that involve complementation are complete for P. It is the combination of concatenation (which is essential for regular expressions) and complement that causes the P-completeness. LTL trades concatenation for complement and, in contrast to regular expressions, provides complementation together with an efficiently parallelizable membership test.

CTL tree checking. Another prominent temporal logic in is *Computation Tree Logic (CTL)*. Analogously to LTL path checking we ask if CTL tree checking can be performed efficiently in parallel. The approach for LTL path checking can indeed be generalized to CTL tree checking. We will prove that CTL tree checking is in $\mathsf{AC}^2(\mathsf{logDCFL}) \subseteq \mathsf{SAC}^3$.

Whereas in the classical domain of model checking, the verification of reactive systems, the use of CTL tree checking appears to be quite limited, there are many other fields where labeled trees are an important data structure that is queried or checked for properties. This includes assertion checking, querying, debugging, and searching in all kinds of parse trees, class hierarchies, thread or process trees, abstract data types, file systems, and XML documents and XML databases. In particular expressiveness and complexity of query languages for XML have received a lot of attention during the last ten years (cf. [64] for pointers to the literature) Probably the most relevant language for querying XML documents is the W3C standard *XML Path Language (XPath)* [90]. It is essential part of other XML techniques like XSLT [89] or XQuery [88]. XPath as well as Core XPath, the navigational fragment of XPath [36], are not complete for FO on unranked trees [73]. The reason is the inability to select an element under the condition that a predicate holds for all locations along the path up the selected



element. This kind of property is closely related to the *until* modality of temporal logics [73]. In [73] Marx defines *conditional XPath (CXPath)* by extending XPath with an correspondent of the temporal *until* operator and shows that CXPath is complete for FO on unranked trees. For temporal logics Barceló and Lipkin prove in [7] that multi-modal CTL* with past is expressively complete for FO on unranked trees. Both logics exhibit the same complexity for unary queries as standard XPath and Core XPath, namely polynomial time completeness [38]. On the other hand, Gottlob, Koch, Pichler, and Segoufin investigate the query complexity of fragments of XPath on unranked trees [38]. In particular they show that query processing for Core XPath without negation is in logCFL.

How does CTL fit into this picture? On one hand, it is clearly not as expressive as multi-modal CTL* with past, not even as expressive as CTL*. Similarly, it is less expressive then CXPath. This is also reflected in the complexity of query processing in that CTL on trees is not hard for polynomial time. On the other hand, its expressiveness is incomparable with XPath, Core XPath and Core XPath without negation. Whereas these languages are similar to multi-modal CTL* in that they provide modalities to navigate on various dimension of the tree (child-relation, parent-relation, and sibling relation, together with the respective transitive closures) and in that they provide means to nest path-expressions and state-predicates, these languages all lack a binary modality comparable to the *until*-operator of CTL and CTL*. From a complexity point of view, our upper bound is better than the bounds for XPath and Core XPath. In certain scenarios this might be a beneficial trade-off: buy a binary until-operator for the restriction to a single modal dimension, namely the child-relation; additionally, gain an efficiently parallel query processing complexity instead of inherently sequential complexity. Our upper bound of SAC^3 is still worse than logCFL for Core XPath without negation. However, the low complexity for negation-free Core XPath comes at a high prize: The lack of negation inhibits any kind of (nested) universal quantification both over branches as well as along a single path.

LTL model checking of classes of Kripke structures. The results in LTL path checking and CTL tree checking show that the model checking problem can be considerably easier for restricted classes of Kripke structures compared to the general case. This motivates to further investigate the properties of Kripke structures that influence the complexity of the model checking problem. The study of the state explosion problem can be seen from this perspective. The state explosion problem occurs in compositional model checking when the Kripke structure is represented as some kind of product Kripke structure. There are different approaches to tackle the state explosion by tuning model checking algorithm to the peculiarities of the product structure. Prominent approaches in this direction are partial order reduction [52, 86, 33] and symmetry reduction [26, 19, 46]. In

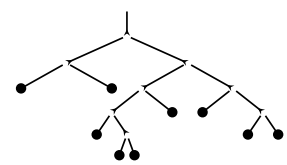
[22] Demri, Laroussinie, and Schnoebelen study the complexity of model checking parameterized by the number of components of the product Kripke structures.

However, somewhat surprisingly, there is almost no research that explicitly considers the frame of the Kripke structure as parameter for the complexity of the model checking problem for LTL and CTL. Aside of the work of Demri, Markey, Raskin, and Schnoebelen on the path checking problem for various temporal logics [23, 71, 72, 70] we are not aware on any further results explicitly considering the complexity of model checking with respect to the frame of the Kripke structure. In the thesis we provide a classification of the complexity of the model checking problem for LTL with respect to the frame of the Kripke structure. For CTL the gap between NC as an upper bound and L as a lower bound for tree structures and P for general structures is comparably small. We derive some corollaries from previously known proofs and point out some open questions.

Path checking for extensions of LTL. The pure logic LTL is rarely used in practice. Practical versions of LTL, such as the IEEE property specification language (PSL) [45], extend the logic with additional operators that help the user to write shorter and simpler specifications. Such extensions often come at a price: adding extended regular expressions, for example, makes the path checking problem P-complete [76]. In the thesis we show that this is not always the case: past-time and bounded operators are two major extensions of LTL, which both improve the succinctness of the logic exponentially, and whose path checking problems remain efficiently parallelizable.

Past-time operators are the dual of the standard modalities, referring to past instead of future events. Past-time operators greatly simplify properties like "b is always preceded by a", which, in the core logic, require an unintuitive application of the Until operator, as in $\mathrm{G}\,\neg(\neg a\,\mathrm{U}\,b \wedge \neg a)$. Furthermore, Laroussinie, Markey and Schnoebelen [63] proved that the property "all future states that agree with the initial state on propositions $p_1, p_2, \ldots p_n$, also agree on proposition p_0," which can obviously be expressed as a simple past-time formula, requires an exponentially larger formula if only future-time operators are allowed.

Bounded operators express that a condition holds *at least* for a given, fixed number of steps, or must occur *within* such a number of steps. Bounded specifications are especially useful in monitoring applications [27], where unbounded modalities are problematic: if only the finite prefix of a computation is visible, it is impossible to falsify an unbounded liveness property or validate an unbounded safety property. The succinctness of the bounded operators is due to the fact that expanding the bounded operators into a formula tree replicates subformulas, causing an exponential blow-up in the formula size. Another exponential blow-up is due to the logarithmic encoding of the bounds compared to an unary encoding in the form of nested next-operators.



A naive solution for the path checking problem of the extended logic would be to simply expand the formula to the core fragment and then apply the construction from Section 4.2. Because of the exponential blow-up, however, such a solution would no longer be in NC. Can we, instead, extend the approach for pure LTL to also handle past and bounds? As we will see, the answer is positive, but requires modifications of the algorithm. The complexity of the path checking problem remains $\mathsf{AC}^1(\mathsf{logDCFL})$ for LTL with past as well as for LTL with bounds.

Related Work. Modern tense logic was founded by Arthur Prior [79]. By adding the *until*- and the *since*-modality to the logic Kamp could prove that it is expressively complete for first order logic on linear orders [50]. Amir Pnueli introduced *Linear-Time Temporal Logic (LTL)* for the verification of computer programs [78]. Sistla and Clarke show in a seminal paper [85] that LTL model checking is PSPACE-complete. However, the data complexity is only linear while the expression complexity is PSPACE [65]. *Computation Tree Logic (CTL)* was introduced by Emerson and Clark in [25]. It has a model checking problem with bilinear complexity [18].

There is a comprehensive line of research that covers all kinds of variations (restrictions and extensions) of the input formula [85, 59, 23, 11, 10] for LTL; for CTL [14, 2] for classical modal logics cf. [44].

LTL path checking was introduced as an open problem by Demri and Schnoebelen in [23]. In [71] Markey and Schnoebelen investigate the path checking problem for various extensions and restrictions of LTL. In [72] Markey and Schnoebelen show that the complexity of the (finite) path checking problem for the μ-calculus is P-hard. In [70] Markey and Raskin study the complexity of the model checking problem for restricted sets of path for extensions of LTL to continuous time.

We are not aware of any research results on the complexity of model checking CTL on trees. However there is a abundant research on query languages and logics to reason about trees. For a general overview refer to [12]; cf. [36, 73, 64, 37, 38, 12, 83] for some research related to temporal logic and XML. Core XPath was introduced in [36]. Its complexity is investigated in [36] and [38]. $\mathcal{C}$XPath, a first order complete extension of XPath, with introduced in [73].

In classical modal logic systems are defined via frame conditions. Starting with Ladners seminal results in [62] there is a line of research about the complexity of problems for modal logics systems under certain frame conditions (cf. [43, 42] for recent results and overview on past work).

The LTL path checking problem is closely related to the membership problems for the various types of regular expressions: the membership problem is in NL for regular expressions [48], in logCFL for semi-extended regular expressions [77], and

P-complete for star-free regular expressions and extended regular expressions [76].

Monitoring LTL is a key problem in runtime verification (cf. [28, 30, 31, 40, 9]). Prominent tools for the synthesis of monitor circuits from PSL are FoCs [20], developed at IBM Haifa, and MBAC by Boulé and Zilic [15]. Our tool [57] is optimized for specification containing bounded subproperties [27]. For temporal logic, an automata-theoretic construction (based on determinization) is due to Armoni et al. [5].

There is a lot of research about real-time temporal logics (cf. [47, 3, 56, 4]) that are interpreted over computations paths where each state is stamped with a value from a real-valued time domain, so called timed state sequences. Here we interpret the bounds over normal computation paths, i.e. the bounds just count states. This approach is common in hardware verification [45] where the system under consideration is assumed to be clocked.

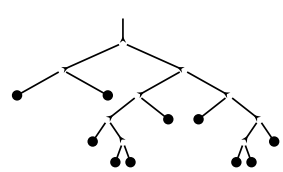
1.2 Boolean Circuit Based Model Checking

Our results on LTL path checking and CTL tree checking for LTL are based on the reduction of the respective model checking problem to the evaluation of monotone planar Boolean circuits. The constructions rely on the following properties of the logics:

Linear positive normal form: It is essential to obtain *monotone* circuits in first place. Since circuits in general are DAGs, the transformation of a circuit into a monotone circuit (that allows negation only on its inputs) is expensive. For temporal logic formulas (which are trees) however it is possible to recursively propagate negations to the level of atomic propositions.

Expansion laws: We obtain a Boolean circuit by expanding the formula over the Kripke structure such that in the resulting structure is the normal product of the parse tree of the formula and the Kripke structure. Assuming that the Kripke structure is acyclic, the product is acyclic as well.

As an example, let us consider the LTL formula $\phi = (a \,\mathrm{U}(b \,\mathrm{U}\, c)) \,\mathrm{U}(d \,\mathrm{U}\, e)$ and a computation path $\rho = \rho_0, \rho_1, \ldots$. We build a circuit $\mathrm{cir}(\phi, \rho)$ by recurring on the subformulas of ϕ and the suffices $\rho_{0,\ldots}, \rho_{1,\ldots}, \ldots$ of ρ. For an atomic proposition p we have that $\mathrm{cir}(p, \rho)$ is just the value of p in the state ρ_0. For an until formula we apply the expansion laws for LTL which results in $\mathrm{cir}(\chi \,\mathrm{U}\, \psi, \rho) = \mathrm{cir}(\psi, \rho) \vee (\mathrm{cir}(\chi, \rho) \wedge \mathrm{cir}(\chi \,\mathrm{U}\, \psi, \rho_{1,\ldots}$ as shown in Figure 1.1 for the top-level U-operator of ϕ. Figure 1.2 shows what we get when iterating the expansion on the suffixes of ρ. Figure 1.3 shows the result of expanding the left-hand operand of the top-level U-operator of ϕ along the suffixes of ρ. The expansion tree is folded



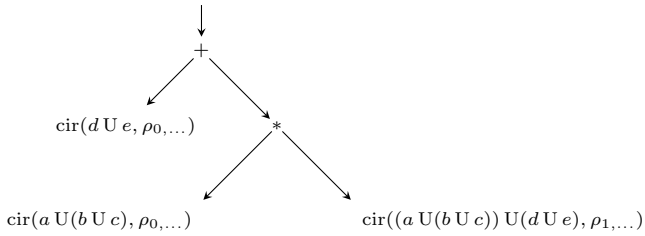


Figure 1.1: The circuit $\mathrm{cir}(\phi, \rho)$ obtained through the expansion of the top-level U-operator of the LTL formula $\phi = (a\,\mathrm{U}(b\,\mathrm{U}\,c))\,\mathrm{U}(d\,\mathrm{U}\,e)$. The symbol $*$ denotes conjunction and $+$ denotes disjunction.

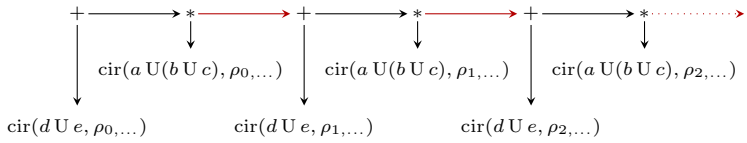


Figure 1.2: Iteration of the expansion from Figure 1.1 along the suffixes of ρ.

into a compact circuit by using dynamic programming. Figure 1.4 unfolds the constants e and d along the suffixes of ρ.

Thus, the problem of checking paths and trees for LTL and CTL, respectively, reduces to the problem of evaluating monotone Boolean circuits. Ladner showed in [61] that the evaluation of Boolean circuit is complete for polynomial time and Goldschlager strengthened this in [34] by proving that the problem is complete for polynomial time even for *monotone* Boolean circuits. The key is to observe that the resulting circuits exhibit a certain topology that allows for a more efficient evaluation. Again, let us focus on the case of LTL. We saw that expanding the example formula ϕ over ρ results in a circuit that is the normal product of ϕ and ρ. Figure 1.5 provides a more global (and schematic) view on the construction: The formula is copied along the path positions and at the same time the path is copied along all subformulas. The important thing to note about this structure is that each directed path in the formula tree corresponds to a planar circuit, i.e. can be projected onto a plane without crossing edges.

This is illustrated in Figure 1.6 that shows the planar circuits for some arbitrary decomposition of the formula tree into directed paths. From this observation it is only a small step to an efficient evaluation of the overall circuit: Although

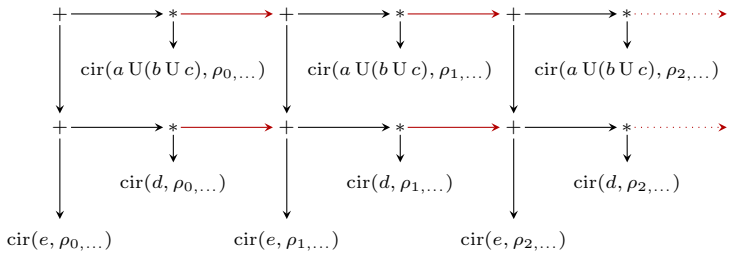


Figure 1.3: Expansion of the left hand operand from the formula in Figure 1.1 along the suffixes of the ρ.

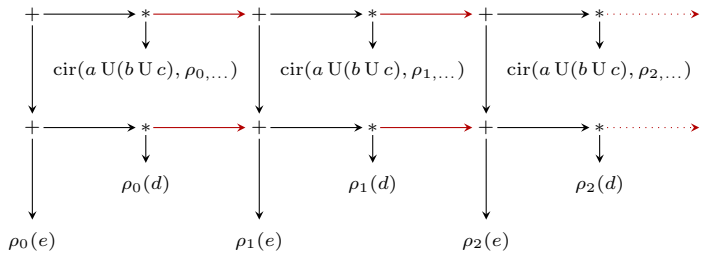
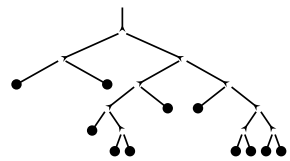


Figure 1.4: Expansion of the constant e and d in the example from Figure 1.1.



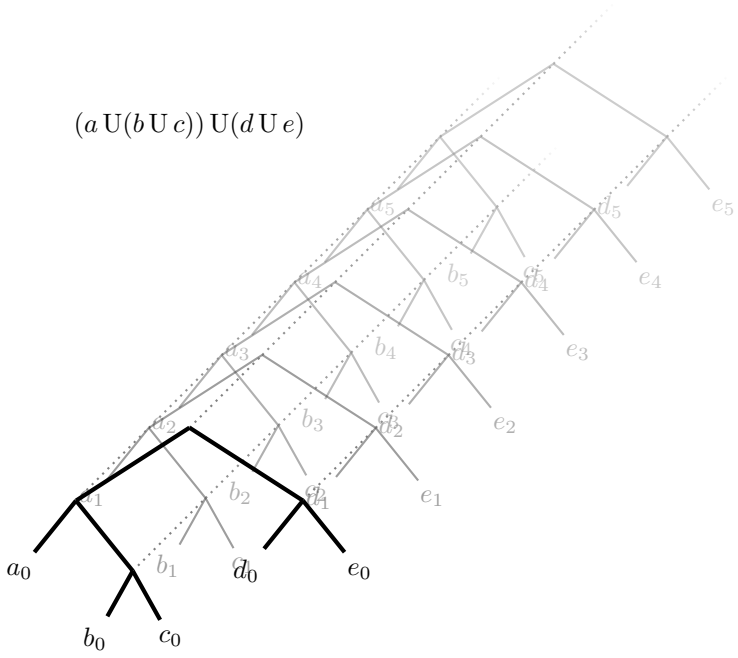


Figure 1.5: Schematic view of the circuit obtained by expanding $\phi = (a\,\mathrm{U}(b\,\mathrm{U}\,c))\,\mathrm{U}(d\,\mathrm{U}\,e)$ over ρ. The graph of the circuit is the normal product of the formula tree of ϕ and the path ρ.

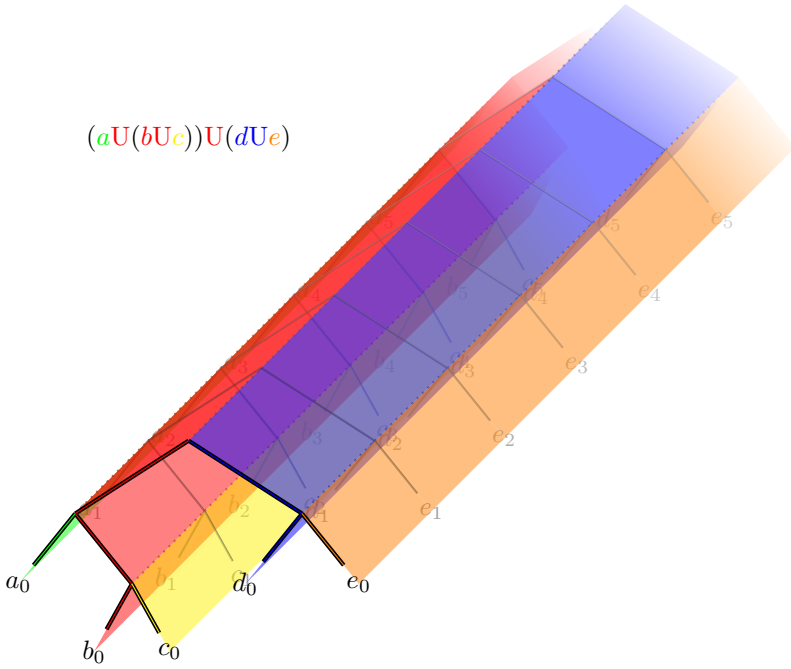
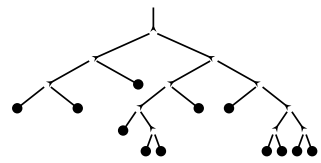


Figure 1.6: Decomposition of the circuit from Figure 1.5 into subcircuits corresponding to directed paths in the formula tree. Observe that each subcircuit of the decomposition is planar.



Goldschalger proved in [34] that the monotone as well as the planar circuit value problems are complete for polynomial time, two years later in [35] he showed that circuits that are both monotone *and* planar can be evaluated in NC under a certain topological restriction. This restriction was dropped independently by Yang in [91] and Delcher and Kosaraju in [21]. Using this result in our example we can evaluate a subcircuit that corresponds to a directed path in the formula tree in NC. If we can decompose the formula tree into directed paths in such a way that the evaluation of the corresponding circuits can be parallelized up to an logarithmic factor then we would get an algorithm that performs the evaluation of the overall circuit in NC. Fortunately, it is well known that for trees such a decomposition always exists for sufficiently cheap and associative evaluation operators [51]. We will use a standard parallel tree contraction algorithm (that is very common e.g. for the evaluation of arithmetic terms) that we piggyback with an appropriate operation for the evaluation and composition of subcircuits. The parallel tree contraction works by *partially evaluating* a node in a tree as soon as one of it children is a leaf (i.e. is fully evaluated). The corresponding leaf is then removed from the tree and simple path are collapsed into a single edge. This is done in parallel for all nodes. Figure 2.1 shows a contraction sequence for the circuit the results from the example formula ϕ.

In the thesis we will formalize this approach and show how it can be extended to circuits that occur in the case of CTL and circuits the result from extensions of LTL with past operators and bounded future operators.

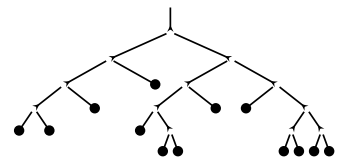
Related Work. The problem of evaluating Boolean circuits has been studied extensively in the literature since Ladner proved the general problem to be complete for polynomial time under logspace reductions [61]. Goldschlager extended this result to monotone Boolean circuits as well as planar Boolean circuits [34]. Two years later he showed that the intersection of both classes, namely, Boolean circuits that are both monotone and planar, can be evaluated in NC^2 under a certain topological restriction: the circuits must be upward-stratified [35]. The upper bound for this class of circuits was improved to logCFL by Dymond and Cook [24] and later by Barrington, Lu, Miltersen, and Skyum to logDCFL [8]. In the meantime Kosaraju relaxed the restriction to upward-stratified circuits to so called focused circuits within a complexity of NC. Finally, Yang [91] and independently Delcher and Kosarja [21] presented NC algorithms for the unrestricted problem of evaluating monotone planar Boolean circuits. Ramachandran and Yang presented algorithms for the restricted version that optimizes the overall size of the circuits [81, 82]. A comprehensive overview about the research on the topic can be found in [67]. The restriction to upward-stratified circuits was recently relaxed by Chakraborty and Datta. They reduce the more general case of monotone planar circuits with all constant gates on a single face to the case of

upward-stratified circuits which makes logDCFL algorithm from [8] applicable for the more general case. Recently, there has been some on sub-polynomial complexities for circuits that are not planar but can be embedded onto the cylindrical and toroidal surfaces [67].

Parallel tree contraction algorithms are well known and used for long time as a algorithmic tool for parallel evaluation algorithms. An overview can be found in [51]. In the thesis we follow their presentation of an approach that goes back to [1] and [55].

1.3 Contributions of the Thesis

- As main result of the thesis we present an efficient parallel algorithm for LTL path checking. The algorithm improves the previously best know upper bound from P to $\mathsf{AC}^1(\mathsf{logDCFL})$. The has been published in *L. Kuhtz and B. Finkbeiner. LTL path checking is efficiently parallelizable. ICALP'09* [58].
- The thesis offers an efficient parallel algorithm for CTL tree checking. The algorithm is in $\mathsf{AC}^2(\mathsf{logDCFL})$ and it establishes the first upper bound that separates the complexity of CTL tree checking from general CTL model checking, which is P-complete.
- LTL with Past-time modalities (LTL+Past) is exponentially more succinct than pure future LTL. The thesis shows that the efficient parallel path checking algorithm for LTL can be extended to LTL+Past. Albeit the compactness of LTL+Past, path checking for LTL+Past is in $\mathsf{AC}^1(\mathsf{logDCFL})$.
- LTL with bounded modalities (BLTL) is another exponentially more succinct extension of LTL. It is of particular relevance for applications of the path checking problem in monitoring and runtime verification. The thesis offers an extension of the efficient parallel path checking algorithm for LTL+Past to BLTL. As for LTL+Past, albeit the compactness of BLTL, the path checking problem is still in $\mathsf{AC}^1(\mathsf{logDCFL})$. In fact, we prove the stronger result that path checking for the combined extension of LTL with bounded modalities and past-time modalities ($BLTL + Past$) is in $\mathsf{AC}(\mathsf{logDCFL})$.
- The path and tree checking algorithms are based on efficient parallel evaluation strategies for monotone Boolean circuits. In the thesis the evaluation of product circuits is reduced to the problem of evaluating one-input-face monotone planar Boolean circuits: for a monotone Boolean circuit that is a product of a tree and a path, an AC^1-reduction is provided; for a monotone Boolean circuit that is a product of two trees, an AC^2-reduction is provided.



- The thesis develops a classification of Kripke structures with respect to the complexity of LTL model checking. By identifying relevant properties of the frame, three main classes of Kripke structures are characterized: Kripke structures for which the problem is PSPACE-complete, Kripke structures for which the problem is coNP-complete, and Kripke structures for which the problem is in NC.

1.4 Organization of the Thesis

The second chapter contains preliminaries. It offers some common notations about graphs and trees; it introduces semantic notions for the logics that we will work with, namely computations and Kripke structures; it presents some basic facts about complexity classes within P and AC-reductions; finally, the parallel tree contraction algorithm is presented. The algorithm an important ingredient in the path and tree checking constructions that are developed in the thesis.

The third chapter of the thesis is devoted to monotone Boolean circuits, as they are the central data structure in most results of the thesis. We will define monotone Boolean circuits in a way that is tailored to our needs. We introduce concepts and notation to conveniently talk about subcircuits, decomposition, and composition of circuits. A particular focus lies on monotone planar Boolean circuits and their evaluation, because this provides us with our most powerful algorithmic tool. We use the results on monotone planar Boolean circuits to derive evaluation strategies for monotone Boolean circuits with a more complex topology.

The fourth chapter is about LTL. It features the main result of the thesis, the efficient parallel path checking algorithm for LTL. Additionally, it investigates the complexity of model checking for classes of Kripke structures with a restricted frame.

The fifth chapter is about CTL. Using the algorithmic techniques that where used already in the fourth chapter about LTL, in this chapter, we apply them to the more complex problem of CTL tree checking.

The sixth chapter deals with extensions of LTL that are of particular relevance for applications of path checking in monitoring in testing. Namely, we provide efficient parallel path checking algorithms for LTL with past-modalities and for LTL with past- and bounded modalities.

The eighth chapter concludes the thesis. It recalls the results and lists some open questions and directions for future work.

Chapter 2

Preliminaries

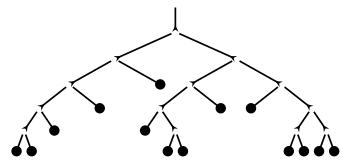
2.1 Directed Graphs and Trees

We introduce some common notation and some convenient abbreviations for directed graphs and trees. The reader is encouraged to skip this part and only refer to it when needed.

A *directed graph* is represented as a tuple $G = \langle V, E \rangle$ where V is the set of vertices and $E \subseteq V \times V$ is the edge set of the graph. If not stated otherwise, we assume V to be finite. For an arbitrary graph G we write $V(G)$ to denote its vertex set and $E(G)$ to denote its edge set. Often we identify a graph with its set of vertices and write V for G. Sometimes, particularly when the edge set is implicit, for $v, w \in V(G)$ we write $\langle v, w \rangle \in G$ instead of $\langle v, w \rangle \in E(G)$ and say the edge $\langle v, w \rangle$ is in the graph G. The in-degree of a vertex $v \in V(G)$ is $|\{\langle w, v \rangle \in E(G) \mid w \in V\}|$. The out-degree of v is $\{\langle v, w \rangle \in E(G) \mid w \in V\}$. The degree of v is the sum of its in-degree and out-degree.

A graph G is *connected* if the symmetric transitive hull of $E(G)$ is $V(G) \times V(G)$. A graph G is *strongly connected* if the transitive hull of $E(G)$ is $V(G) \times V(G)$. A (directed) *path* of length n is a connected graph with n vertices, with $n - 1$ edges and each vertex has in-degree and out-degree at most one. For a path the unique vertex with in-degree of zero (out-degree of zero) is called the start-vertex (end-vertex) of the path. A (directed) *cycle* is a connected graph with in-degree and out-degree of exactly one for each vertex. The *cycle-graph* of a graph G is the graph that is obtained from G by collapsing each cycle in G into a single vertex.

A graph G is a *subgraph* of a graph H, denoted as $G \subseteq H$ if $V(G) \subseteq V(H)$ and $E(G) \subseteq E(H)$. We call G a *spanning subgraph* of H if $G \subseteq H$ and $V(G) = V(H)$. We say that G is an *induced subgraph* of H if $G \subseteq H$ and $E(G) = E(H) \cap (V(G) \times$



$V(G)$). For $G \subseteq H$ we say that G is a path (cycle, tree, etc.) *in* H if the induced subgraph on $V(G)$ is a path (cycle, tree, etc.). Usually, we consider $\subseteq$ modulo graph-isomorphism, i.e. under adequate renaming of vertices. A subgraph $H \subseteq G$ is called *convex* in G if for each pair of vertices $\langle v, w\rangle \in H$ any path in G that starts in v and ends in w is also a path in H. For a graph G and two vertices $v, w \in G$ we say that w is *reachable* from w if there is a path in G that starts with v and ends with w.

A (rooted) *tree* is a connected graph $T = \langle V, E\rangle$ where for each node v there is a unique node w with $\langle w, v\rangle \in E$ except for a single node, the *root node* of T, that has no predecessor in E and is denoted as root(T). We also call the vertices of a tree *nodes*. For any two nodes $v, w \in V$ there is at most one path $P \subseteq V$ that starts in v and ends in w. We say that a node v is a *child node* of a node w if $\langle w, v\rangle \in E$; w is called the *parent node* of v. A node v is called a *sibling node* of a node w if they have the same parent node. A node without children is called a *leaf node*. The subgraph that is induced by a node v along with all nodes that are reachable from v is called the *subtree* rooted at v. The subtree rooted at a node v itself is a tree. For a node v we call the subtree rooted at child node of v a *child-tree* of the subtree rooted at v. The *degree* of a tree is the maximum number of children of a node in the tree. We call a tree *regular* if for each non-leaf node the number of children equals the degree of the tree.

Given two graphs G and H, the *normal product* $G \boxtimes H$ of G and H is the graph I with $V(I) = V(G) \times V(H)$ and $\langle\langle g_0, h_0\rangle, \langle g_1, h_1\rangle\rangle \in E(I)$ if and only if

$$\langle g_0, g_1\rangle \in E(G) \wedge h_0 = h_1, \text{ or}$$
$$g_0 = g_1 \wedge \langle h_0, h_1\rangle \in E(H), \text{ or}$$
$$\langle g_0, g_1\rangle \in E(G) \wedge \langle h_0, h_1\rangle \in E(H).$$

2.2 Computation Paths, Computation Trees, and Kripke Structures

Temporal logics come in two flavors: *linear-time temporal logics* and *branching-time temporal logics*. Linear-time logics reason about linear ordered sequences of states, which we call computation paths in this thesis. Branching-time logics reason about (rooted) trees of states, which we call computation trees, respectively. In the course of the thesis we consider two prominent temporal logics. Linear time temporal logic (LTL) and computation tree logic (CTL), a branching-time logic.

In this section we define the semantic framework for these logics: propositions, states, computation paths, computation trees, and Kripke structures. Kripke structures are a unified framework for symbolically representing both computa-

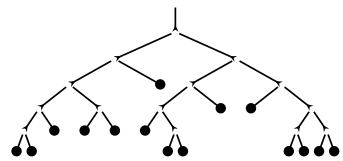
tion paths and computation trees in a single structure. We will conclude this section by defining the model checking problem on Kripke structures in general and classes of Kripke structures in particular.

Given a set of *atomic propositions* AP. A *state* $s \in 2^{\mathrm{AP}}$ is an evaluation of the atomic propositions in AP. For $p \in \mathrm{AP}$ we say that p holds in s if and only if $p \in s$. We write $s(p)$ to denote the value of p in s with $s(p) = 1$, if p holds in s, and $s(p) = 0$ otherwise. An ordered sequence $\rho = \rho_0, \rho_1, \ldots$ of states is called a *computation path* over AP. The *length of* ρ is denoted by $|\rho|$. If ρ is infinite, we set $|\rho| = \infty$; $i < \infty$ for all $i \in \mathbb{N}$. For a computation path ρ and $0 \leq i < |\rho|$ we write ρ_i for the state at position i; $\rho_{i,j}$, where $0 \leq i \leq j < |\rho|$, denotes the computation path $\rho_i, \rho_{i+1}, \ldots, \rho_j$ of length $|\rho_{i,j}| = j - i + 1$; $\rho_{i,\ldots}$ denotes the suffix of ρ at position i. The empty sequence is denoted ϵ with $|\epsilon| = 0$. We denote concatenation of computation paths as a product and write either $\sigma\rho$ or $\sigma \cdot \rho$ for the concatenation of the computation paths σ and ρ, where σ is finite. For a finite computation path σ we set $\sigma^n = \prod_0^{n-1} \sigma$, $\sigma^* = \left\{ \prod_0^{n-1} \sigma \mid n \in \mathbb{N} \right\}$, and $\sigma^\omega = \prod_0^{\infty} \sigma$. In the context of automata we will treat computation paths over AP as *words* over the *alphabet* $\Sigma = 2^{\mathrm{AP}}$, where a *letter* is a state. The set of all finite words over Σ is denoted as Σ^*. The set of infinite words is denoted as Σ^ω. A *language* over Σ is a subset of $\Sigma^* \cup \Sigma^\omega$. A computation path (or a word) $\rho = \rho_0, \rho_1, \ldots, \rho_n, n \in \mathbb{N}$ canonically defines a path. In the following we view ρ as a path whenever adequate. Similarly to computation paths, we define *computation trees*. A computation tree over a set of atomic propositions AP is a finite or infinite tree where the nodes are labeled with subsets of 2^{AP}. The empty computation tree is denoted as ϵ.

A *Kripke structure* $\mathcal{K}$ is a four-tuple $\langle K, k_i, R, \lambda \rangle$ where K is a set of vertices, $k_i \subseteq K$ are the initial vertices, $R \subseteq K \times K$ is a transition relation, and $\lambda \colon K \to 2^{\mathrm{AP}}$ is a labeling function on the vertices of $\mathcal{K}$. By abuse of notation we sometimes identify a state $k \in K$ with its labeling $\lambda(k)$, where we assume that $\lambda^{-1}(k)$ is determined from the context. The *language of a Kripke structure* $\mathcal{K} = \langle K, k_i, R, \lambda \rangle$, denoted as $\mathrm{lang}(\mathcal{K})$, is the set of (finite and infinite) computation paths $\{\lambda(s_0), \lambda(s_1), \cdots \mid s_0 \in k_i, \langle s_i, s_{i+1} \rangle \in R\}$ for $i \in \mathbb{N}$ with $0 \leq i$ or $0 \leq i < n$ for some $n \in \mathbb{N}$. Let T be a finite or infinite tree. Let $\mu \colon V(T) \to K$ be a labeling of T such that $\mu(\mathrm{root}(T)) \in k_i$ and for each $\langle s, t \rangle \in E(T)$ it holds that $\langle \mu(s), \mu(t) \rangle \in R$. Then the tree τ with

$$V(\tau) = \{\mu(v) \mid v \in T\} \text{ and}$$
$$E(\tau) = \{\langle \mu(v)), \mu(w) \rangle \mid \langle v, w \rangle \in E(T)\}$$

is called a computation tree of $\mathcal{K}$ with labeling λ. The *tree language of a Kripke structure* $\mathcal{K}$, denoted as $\mathrm{lang}^\tau(\mathcal{K})$ is the set of finite and infinite computation trees of $\mathcal{K}$. A finite computation path, respectively a finite computation tree, can be interpreted as a Kripke structure itself.



Given a linear-time temporal logic $\mathcal{L}$ with a satisfaction relation $\models$ for computation paths. For formula $\phi \in \mathcal{L}$ and a Kripke structure $\mathcal{K}$ we say that $\mathcal{K}$ satisfies ϕ, denoted as $\mathcal{K} \models \phi$ if and only if for each computation path ρ in lang($\mathcal{K}$) it holds that $\rho \models \phi$. Similarly, for a branching-time temporal logic $\mathcal{L}$ with satisfaction relation $\models$ for computation trees we say that a Kripke structure satisfies a formula $\phi \in \mathcal{L}$ if and only if for all computation trees τ in lang$^{\tau}(\mathcal{K})$ it holds that $\tau \models \phi$.

Given a class of Kripke structures $\mathcal{K}$ and a temporal logic $\mathcal{L}$. The *model checking problem* of $\mathcal{L}$ over $\mathcal{K}$ (MC[$\mathcal{L}$, $\mathcal{K}$]) is the defined by

$$\mathrm{MC}[\mathcal{L}, \mathcal{K}] = \left\{ K \stackrel{?}{\models} \phi \mid K \in \mathcal{K}, \phi \in \mathcal{L} \right\} \quad .$$

2.3 Complexity Classes in P

This section provides some preliminaries about computational models and complexity classes for problems that are contained in P, i.e. problems that can be solved by using at most a polynomial number of sequential atomic computation steps. The presented content is and can be found in any standard textbook on complexity theory (see e.g. [75]). Nevertheless, we recall some basic notions here, as complexities below P are rare in the context of temporal logic model checking.

A Boolean circuit is a directed acyclic graph where the vertices (called gates) are labeled with Boolean connectives and the edges (called dependencies) bind an operand of a gate to the output of another gate. The input degree of a gate is called fan-in. The fan-out of a gate is its output degree. A monotone Boolean circuit is a Boolean circuit where negations are allowed to occur only at the input level and all remaining gates are and-gates or or-gates. For a problem P and a complexity class C we say that P is decided by a C-uniform family of circuits if there is an algorithm in C that computes for each $i \in \mathbb{N}$ a circuit with input length i that decides all instances of P of length i. Throughout the thesis we only consider logspace-uniform families of circuits and only uniform complexity classes.

We do not explicitly distinguish between decision problems and functional problems. Since in our case the output size of functions is always polynomially bounded we can use a polynomial number of circuits for the corresponding class of decision problems, each for computing a single bit of the output [49].

In the following we provide a brief overview about the most prominent complexity classes within P. NL is the class of problems that can be decided by a logspace restricted (non-deterministic) Turing machine. L is the class of problems

that can be decided by a logspace restricted deterministic Turing machine. logCFL is the class of problems that can be decided by a logspace and polynomial time restricted nondeterministic Turing machine that is additionally equipped with a push-down stack. It is equivalent to the class of problems that are L-reducible to a context-free language. Moreover, it is equivalent to the class SAC^1 of problems decidable by a uniform family of polynomial size monotone Boolean circuits of logarithmic depth with constantly bounded fan-in either for all and-gates or for all or-gates. logDCFL is the class of problems that can be decided by a logspace and polynomial time restricted *deterministic* Turing machine that is additionally equipped with a push-down stack. It is equivalent to the class of problems L-reducible to a deterministic context-free language. $\mathsf{AC}^i, i \in \mathbb{N}$, denotes the class of problems decidable by polynomial size unbounded fan-in Boolean circuits of depth $\log^i$. $\mathsf{NC}^i, i \in \mathbb{N}$, denotes the class of problems decidable by polynomial size bounded fan-in Boolean circuits of depth $\log^i$. NC is the set of decision problems decidable in poly-logarithmic time on a parallel computer (PRAM) with a polynomial number of processors. It holds that $\mathsf{NC} = \bigcup_{i\in\mathbb{N}} \mathsf{NC}^i = \bigcup_{i\in\mathbb{N}} \mathsf{AC}^i$. The following inclusions are known:

$$\mathsf{AC}^0 \subsetneq \mathsf{NC}^1 \subseteq \mathsf{L} \begin{array}{c} \subsetneq \\ \subseteq \end{array} \begin{array}{c} \mathsf{NL} \\ \mathsf{logDCFL} \end{array} \begin{array}{c} \subseteq \\ \subsetneq \end{array} \mathsf{logCFL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathsf{AC}^2 \subseteq \cdots \subseteq \mathsf{NC} \subseteq \mathsf{P} \ .$$

Further details can, for example, be found in the survey paper by Johnson [49].

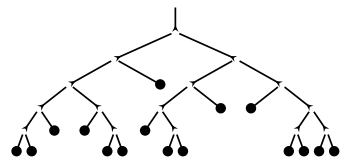
Given a problem P and a complexity class C, P is *C-many-one reducible* to a problem Q if there is an algorithm in C that maps each P-instance p to a Q-instance q such that $p \in P$ iff $q \in Q$. Given a problem P and complexity classes C and D, P is *D-Turing reducible* to C if P can be decided with an algorithm in C that has access to a D oracle where the algorithm can make an oracle call in each computation step. Particularly, given a problem P and a complexity class C, for $n \in \mathbb{N}$ the problem P is AC^n-Turing reducible to C (denoted as $P \in \mathsf{AC}^n(C)$) if there is a family of AC^n circuits with additional unbounded fan-in C-oracle gates that decides P. It holds that

$$\mathsf{AC}^1 \subseteq \mathsf{AC}^1(\mathsf{logDCFL}) \subseteq \mathsf{SAC}^2 \subseteq \mathsf{AC}^2 \ .$$

For further details on AC reductions, we refer to [87].

2.4 Parallel Tree Contraction

The parallel model checking algorithms that are presented in this thesis rely on efficient parallel tree contraction. The approach is based on Abrahamson, Dadoun, Kirkpatrick, and Przytycka [1] and Kosaraju and Delcher [55]. In the



presentation we follow [51]. Let $\mathcal{T}_0 = \langle V_0, E_0 \rangle$ be a binary regular tree. A contraction step on $\mathcal{T}_i = \langle V_i, E_i \rangle$ takes a leaf l of $\mathcal{T}_i$, its sibling s, and its parent p and contracts these nodes into a single node s in the tree $\mathcal{T}_{i+1} = \langle V_{i+1}, E_{i+1} \rangle$ with

$$V_{i+1} = (V_i \setminus \{l, p\}), \text{ and}$$
$$E_{i+1} = \begin{cases} E_i \setminus \{\langle p, l\rangle, \langle p, s\rangle\} & \text{if } p = \text{root}(\mathcal{T}_i), \\ (E_i \setminus \{\langle p, l\rangle, \langle p, s\rangle, \langle pp, p\rangle\}) \cup \{\langle pp, s\rangle\} & \text{otherwise} \end{cases}$$

where $\langle pp, p\rangle \in E_i$. Using the fact that a contraction step is a local operation it is possible to perform contraction steps in parallel on non-overlapping subtrees. A tree contraction on a regular binary tree T is a process that iteratively applies contraction steps on the tree T until it is contracted into a singleton tree. Algorithm 1 from [51] performs a tree contraction in $\lceil \log n \rceil$ stages of parallel contraction steps.

Algorithm 1 Parallel Tree Contraction

Input: a regular binary tree T with n leaves.
Effect: contracts T into a singleton tree.

Number the leaves in order from left to right as $1, \ldots, n$.
for $\lceil \log n \rceil$ iterations **do**
  Apply the contraction step to all odd numbered leaves that are the left child of their parent.
  Apply the contraction step to all odd numbered leaves that are the right child of their parent.
  Shift out the rightmost bit in the numbers of the remaining leaves.
end for

The algorithm can be implemented on an parallel computer (EREW PRAM) such that it runs in time $O(\log n)$ with a total work of $O(n)$ [51]. It is well known that problems that can be solved on an EREW PRAM in time $O(\log n)$ with polynomial total work are contained in AC^1 [87]. Figure 2.1 shows a tree contraction process for an example tree.

In order to use the parallel tree contraction algorithm to compute some function on a labeled tree, the contraction step is piggybacked with a local operation on the labels of the node involved in the contraction step. The complexity of AC^1 for the contraction process assumes that a contraction step is performed in $O(1)$. For our constructions this is not the case. However, by piggybacking the contraction step with C-oracle gates, the tree contraction problem is AC^1-reduced to C. Hence, by showing that the complexity of the contraction step is C, the overall complexity of the contraction algorithm is proved to be $\mathsf{AC}^1(C)$.

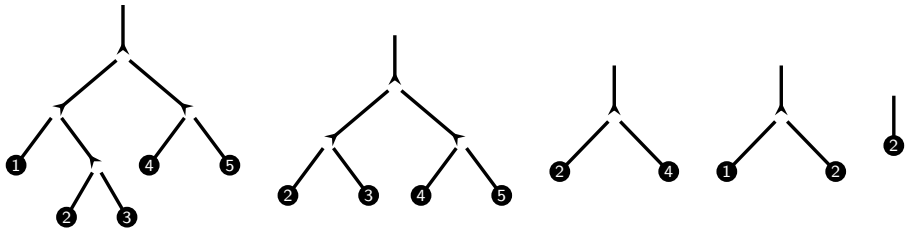
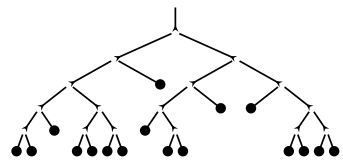


Figure 2.1: A parallel contraction process as produced by Algorithm 1.

Remark: It is straightforward to extend the parallel tree contraction to trees of constantly bounded degree or even arbitrary (rooted and finite) trees. In order keep the presentation simple we restrict ourself to binary regular trees.



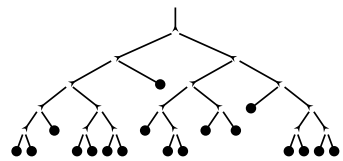
Chapter 3

Monotone Boolean Circuits

The model checking techniques that we present in this thesis make use of monotone Boolean circuits as a central data structure. In this chapter we refine the notion of a Boolean circuit as introduced in the previous chapter. The definition in Section 2.3 was sufficient for the definition of the complexity classes and complexity theoretic reductions. When used as an algorithmic tool for solving model checking problems we need a more elaborate notion of circuits.

An n-ary Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}$ is called *monotone* if for two input vectors a and b with $a \leq b$ (point-wise less or equal) it holds that $f(a) \leq f(b)$. A *monotone Boolean circuit* $\langle \Gamma, \gamma \rangle$ consists of a set Γ of *gates* and a *gate labeling* γ. The gate labeling labels each gate with a tuple $\langle f, \langle g_i \rangle_{i \in I} \rangle$, $g_i \in \Gamma$, containing a monotone Boolean function f of arity n and list of n gates naming the input parameters of f for some index set I. Beside the constant Boolean functions 1 and 0 we introduce an uninterpreted function symbol "?", which we use to model circuits with (variable) input, i.e. circuits that compute a function. Usually, for constant functions we simply write the constant as a label. Moreover, we assume that the list of parameters contains only those parameters that are relevant for the outcome of the function. In particular we require the constant gates to have fan-in 0 and identity gates to have fan-in 1. A gate that is labeled with a Boolean value is called a *constant gate*. A gate that is labeled with ? is called a *variable gate*. For a gate g labeled with $\langle f, \langle g_i \rangle_{i \in I} \rangle$ we say that g *directly depends* on g_i, denoted by $g \succ g_i$. The *dependence* relation ($\succ$) is the transitive closure of $\succ$. A gate on which no other gate depends is called a *sink gate*. *A circuit must not contain any cyclic dependencies.*

Let $C = \langle \Gamma, \gamma \rangle$ be a monotone Boolean circuit. The *graph* of C is defined as $\mathrm{graph}(C) = \langle \Gamma, \succ \rangle$. Clearly, $\mathrm{graph}(C)$ (together with γ) is a (node labeled) acyclic directed graph. For sets of gates $G, G' \subseteq \Gamma$, a gate $a \in \Gamma$, and $* \in \{\succ, \succ\}$



we define

- $G * a$ iff there is a gate $g' \in G$ such that $g' * a$,
- $a * G$ iff there is a gate $g' \in G$ such that $a * g'$,
- $G \succ G'$ iff there is a gate $g \in G$ such that $g \succ G'$ and there is no gate in $g' \in G'$ such that $g' \succ G$.
- $G \succcurlyeq G'$ iff $G \succ G'$ and the subgraph induced on $\mathrm{graph}(C)$ by $G \cup G'$ is convex.

For a circuit $C = \langle \Gamma, \gamma \rangle$, $\mathrm{const}(C)$ denotes the set of all constant gates in Γ. If $\Gamma = \mathrm{const}(C)$, we call C *constant*. By $\mathrm{var}(C)$ we denote the set of all variable gates of Γ. Finally we define $\mathrm{src}(C)$ to be the set of all variable gates and all constant gates that are not sink gates in Γ.

In the following, we assume that all circuits are monotone Boolean circuits. We omit the labeling whenever it is clear from the context and identify the circuit with its set of gates and write Γ for $\langle \Gamma, \gamma \rangle$. Similarly, we often identify the circuit $\langle \Gamma, \gamma \rangle$ and its graph $\mathrm{graph}(\Gamma)$ and write Γ for both.

3.1 Circuit Evaluation

A circuit is evaluated by propagating all constants "upwards" (against the direction of the edges) through the circuit. In a first step we define what it means to evaluate a single gate where possibly some of its dependencies are constants. Intuitively, the function of the gate is partially evaluated by binding it to all constant inputs and removing the corresponding dependencies from the parameter list. Given a circuit $\langle \Gamma, \gamma \rangle$ and a gate $g \in \Gamma$ with $\gamma(g) = \langle f, \langle p_i \rangle_{i \in I} \rangle$, $f \colon \mathbb{B}^{|I|} \to \mathbb{B}$, $\langle p_i \rangle_{i \in I} \in \Gamma^I$ for some index set I. The local evaluation of g in Γ, denoted as $\mathrm{eval}_g(\Gamma)$, is the circuit $\langle \Gamma, \gamma' \rangle$ with

$$\gamma'(x) = \begin{cases} \left\langle f', \langle p_j \rangle_{j \in J} \right\rangle & \text{if } x = g, \\ \gamma(x) & \text{otherwise,} \end{cases}$$

where

$$J = \{ i \in I \mid \gamma(p_i) \notin \{0, 1\} \} \subseteq I, \text{ and}$$
$$f' \colon \mathbb{B}^{|J|} \to \mathbb{B},$$
$$f' = \lambda (x_j)_{j \in J} . f(y_i)_{i \in I}, \text{ with } y_i = \begin{cases} 0 & \text{if } \gamma(p_i) = 0, \\ 1 & \text{if } \gamma(p_i) = 1, \text{ and} \\ x_i & \text{otherwise.} \end{cases}$$

We treat equality between functions as semantic equality. For $|J|$ bounded by a constant c, the number of monotone Boolean functions is constantly bounded by the c^{th} Dedekind number. Given $\gamma'(p_i)_{i\in I}$, $\gamma'(g)$ can thus be computed from $\gamma(g)$ in $O(1)$.

For a set of gates $G \subseteq \Gamma$, let $>$ be a linear order on Γ that is consistent with $\succ$, i.e. $\succ \subseteq >$. We define $\mathrm{eval}_G = \mathrm{eval}_{g_0} \circ \cdots \circ \mathrm{eval}_{g_{|G|-1}}$ with $g_i, g_j \in G$ and $g_j > g_i$ for $i < j$. The *evaluation* of Γ is defined by $\mathrm{eval}(\Gamma) = \mathrm{eval}_\Gamma(\Gamma)$. Propagation of constants is defined similarly, however, the order of the single propagation steps is reversed with respect to the dependency between gates. Let $H = \{h \mid h \succ G\}$ be the set of all gates that depend on G. In contrast to evaluation we define $\mathrm{propagate}_G = \mathrm{eval}_{h_0} \circ \cdots \circ \mathrm{eval}_{h_{|H|-1}}$ with $h_i, h_j \in H$ and $h_i > h_j$ for $i < j$. The reverse ordering of the evaluation steps guarantees that no step depends on the result of a previous step. Hence, we can compute $\mathrm{propagate}_G$ in AC^0 by updating the labeling for all gates (independently) in parallel. For evaluation this is generally not true. We say that a circuit is *evaluated* if all constant gates are sink gates.

Lemma 1. *Given a circuit* Γ*. In* $\mathrm{graph}(\mathrm{eval}(\Gamma))$ *all constant gates are isolated vertices.* □

In an evaluated circuit, all gates that do not depend on variable gates are constant. Hence, a circuit without any variable gates evaluates to a constant circuit; for a circuit that contains variable gates, a subset of the gates is relabeled such that the arity of the functions on the labels decreases. In particular during evaluation edges are never added to the graph of the circuit but only removed from it.

Lemma 2. *Given a circuit* Γ*. It holds that* $\mathrm{graph}(\mathrm{eval}(\Gamma))$ *is a spanning subgraph of* $\mathrm{graph}(\Gamma)$*.* □

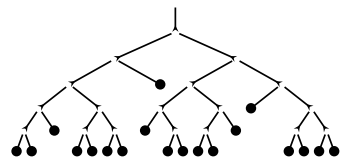
In the following we define circuit equivalence based on circuit evaluation. We start by stating that evaluation is idempotent.

Lemma 3. *For a circuit* Γ *it holds that* $\mathrm{eval}(\Gamma) = \mathrm{eval}(\mathrm{eval}(\Gamma))$*.* □

We say that two circuits Γ and Δ are equivalent, denoted as $\Gamma \equiv \Delta$, if $\mathrm{eval}(\Gamma) = \mathrm{eval}(\Delta)$.

Lemma 4. *For a circuit* Γ *and any set of gates* $G \subseteq \Gamma$ *it holds that*

$$\mathrm{eval}_G(\Gamma) \equiv \mathrm{propagate}_G(\Gamma) \equiv \Gamma$$ □



3.2 Subcircuits, Decomposition, and Composition of Circuits

Given a circuit $\langle \Gamma, \gamma \rangle$, a circuit $\langle \Gamma', \gamma' \rangle$ is *subcircuit* of Γ, denoted by $\Gamma' \sqsubseteq \Gamma$ if and only if $\Gamma' = P \cup \{g \mid P \succ g\}$ such that P is a convex subset of Γ and

$$\gamma'(g) = \begin{cases} \gamma(g) & \text{if } g \in P, \\ ? & \text{otherwise.} \end{cases}$$

By abuse of notation, we often write P for Γ' and $P \sqsubseteq \Gamma$ where we mean $\Gamma' \sqsubseteq \Gamma$. We are careful to do so only if it eases the presentation and does not lead to confusion.

Let $\equiv$ be an equivalence relation on Γ. We call the partitioning $\Gamma/\equiv$ of Γ a circuit *decomposition* of Γ if each equivalence class $[g]_\equiv$ is a subcircuit of Γ ($[g]_\equiv \sqsubseteq \Gamma$)[1]. The dependency $\succ$ on subcircuits induces a partial order on $\Gamma/\equiv$.

Given two circuits $\langle \Gamma, \gamma \rangle$ and $\langle \Delta, \delta \rangle$. A *binding* is a mapping $\beta \colon \Delta' \to \Gamma$, with $\Delta' \subseteq \mathrm{var}(\Delta)$. The *composition* of Γ and Δ under the binding β, denoted as $\Gamma \circ_\beta \Delta$, is a circuit $\langle \mathsf{E}, \varepsilon \rangle$ with $\mathsf{E} = \Gamma \cup (\Delta \setminus \Delta')$ and

$$\varepsilon(g) = \begin{cases} \gamma(g) & \text{if } g \in \Gamma, \\ \delta(g)[d \mapsto \beta(d), d \in \Delta'] & \text{if } g \in \Delta \setminus \Delta'. \end{cases}$$

We define $\circ = \circ_{\mathrm{id}}$ and assume adequate renaming of the gates in the resulting circuit in order to guarantee uniqueness of identifiers. From the definition it follows that circuit composition is the (left) inverse operation of circuit decomposition.

Lemma 5. *Given a circuit $\langle \Gamma, \gamma \rangle$. For a decomposition Γ/R it holds that $\circ_{r \in \Gamma/R}\, r = \Gamma$.* □

Circuit composition does not commute. Therefore the notation $\circ_{r \in \Gamma/R}$ assumes a linear ordering on the elements of Γ/R that has to be consistent with the partial order induced by the subcircuit dependency. In the following we always assume that $\circ$ is applied according to this order.

In the next sections we develop techniques for the evaluation of circuits using a divide and conquer approach: We decompose a circuit into simpler circuits and stepwise evaluate the circuit while recomposing it from its parts. This is justified by the following lemma:

Lemma 6. *Given a circuit $\langle \Gamma, \gamma \rangle$. For a decomposition Γ/R it holds that*

$$\circ_{r \in \Gamma/R}\ \mathrm{eval}(r) \equiv \Gamma. \qquad \square$$

[1] By abuse of notation, instead of $[g]_\equiv \sqsubseteq \Gamma$ we actually mean $[g]_\equiv \cup \{g' \mid [g]_\equiv \succ g'\} \sqsubseteq \Gamma$, as already mentioned.

Circuit composition is associative. This is essential for being able to parallelize the process of evaluation and recomposition.

Lemma 7 (Associativity of $\circ$)**.** *Given circuits P, Q, and R. It holds that*

$$(P \circ Q) \circ R = P \circ (Q \circ R)\,. \qquad \square$$

3.3 Monotone Planar Circuit Value Problem

The problem of evaluating Boolean circuits is P-complete [61]. This also holds for monotone Boolean circuits as well as for planar Boolean circuits [34]. The situation changes if a Boolean circuit is both monotone and planar. In this case evaluation is possible in NC. A first NC^2 upper bound was shown for a special case of the problem in [35]. For general monotone planar Boolean circuits upper bounds in NC were first established independently in [91] and [21].

The model checking algorithms that we present in this thesis are based on the evaluation of monotone one-input-face planar circuits. Remember that we generally assume all circuits to be monotone. The results cited in this section were originally stated for circuits of fan-in two. It is easy to see that the results on planar circuits generalize to the case of unbounded fan-in.

A circuit is *planar* if there exists a planar embedding of its graph. A planar circuit Γ is *one-input-face* if there is a planar embedding such that all gates in $\mathrm{src}(\Gamma)$ are located on the outer face. In the following, we abbreviate *one-input-face planar* as OIF, using the term OIF for the circuits as well as for the corresponding property of a circuit. An evaluated circuit with all variable gates on the outer face is OIF. The evaluation of OIF circuits can be parallelized efficiently. The best currently known algorithm is based on an algorithm from Barrington, Lu, Miltersen, and Skyum [8]. However, their algorithm works only for a specific class (or encoding) of OIF circuits, namely so called upward-stratified circuits. In these circuits the gate set is "layered" such that all dependencies go from a layer to the next lower layer and all input gates are on the bottom layer.

Theorem 1 (Barrington, Lu, Miltersen, and Skyum)**.** *The problem of evaluating* OIF *upward-stratified circuits without variable gates is in* logDCFL.

The basic idea behind the algorithm goes back to a logCFL upper bound from Dymond and Cook [24]. Dymond and Cook observe that in a upward stratified monotone planar circuit the intervals of 1-gates in the bottom (input-) layer are propagated from the lower layers to the upper layers until possibly reaching the top level (output) gate. On a propagation step from one layer to the next upper layer two intervals can merge but due to the monotonicity intervals can never split. This way the propagation of the intervals of 1-gates on the input

layer upward through the circuit yields a tree (or a forest, if we also count trees that do not reach the top level gate) of intervals of 1-gates that certifies the value of the top level gate. The algorithm from Dymond and Cook guesses and checks this tree in a depth-first-search manner which can be done with a logarithmic space restricted and polynomial time restricted Turing machine with an additional pushdown store (logCFL). (The fact that the intervals form a proof *tree* is essential for the polynomial time restriction.) Barrington et al. observe in [8] that this algorithm can be made deterministic by building the proof tree in a bottom-up manner. It selectively propagates adjacent pairs of intervals of 1-gates from the bottom layer upwards through the circuit. If the intervals merge then the algorithm treats them in the following steps as one single interval by flipping the separating 0-gates to 1-gates. Thus the new interval actually represents a whole subtree of the final proof tree. The algorithm uses a pushdown stack to keep track of this intermediate results. We omit the details here and just remark that the fact that intervals can not split guarantees that the algorithm terminates in polynomial time. Since the algorithm is deterministic it is in logDCFL.

Each planar circuit can be turned into an upward stratified circuit. Although layering the nodes might involve the introduction of new identity gates the overall blow-up is polynomial. Chakraborty and Datta show in [17] that this transformation can be done in logDCFL. Although it is overkill – since all circuits that appear in the thesis can easily be made upward stratified in L– we just use the general reduction here:

Theorem 2 (Chakraborty and Datta 2006)**.** *The problem of evaluating* OIF *circuits without variable gates is in* logDCFL*.*

Using standard techniques [54], the algorithm from [8] generalizes to circuits that contain variable gates (on the outer face):

Corollary 1. *The problem of evaluating* OIF *circuits is in* logDCFL*.*

Proof. We first assign the Boolean constant 1 to all variable gates. Each gate that evaluates to 0 is turned into a 0 constant gate. Next, we assign 0 to all variable gates. Each gate that evaluates to 1 is turned into a constant gate with value 1. Since the values of the remaining gates depend on the variables, they are simply copied. If one of the latter gates depends on a constant gate, the dependency is removed by changing such a gate into an *id*-gate. □

3.4 Evaluation of Tree Product Circuits

In this section we develop a reduction from the evaluation of circuits with a certain topology to the problem of evaluating one-input-face planar circuits. The main

idea is to decompose a circuit into a tree of subcircuits and then evaluate and recompose the subcircuits bottom up using the parallel tree contraction algorithm Algorithm 1 from Section 2.4. If needed, this reduction has to be repeated until we get one-input-face planar circuits as base case.

Given a circuit $\langle \Gamma, \gamma \rangle$ without variable gates such that the graph of Γ is a spanning subgraph of the normal product $G \boxtimes H$ of two directed acyclic graphs G and H. The projection on the G-component induces an equivalence relation $\equiv_G$ on Γ. The edges of G induce a partial ordering on the equivalence classes that is a superset of the subcircuit dependency $\succ$ on the partitions of $\equiv_G$. We thus get a circuit decomposition $\Gamma/\equiv_G$.

We will show that, if G is a tree, we can use Algorithm 1 on the tree $\Gamma/\equiv_G$ in order to AC^1-reduce the evaluation of Γ to the evaluation of variable free circuits that are subgraphs of the normal product of a path in G with H. As a special case, if H is a path, then the evaluation of Γ is reduced the problem of evaluating OIF monotone Boolean circuits with variables gates.

We keep the presentation simple and restrict ourself to the case of binary trees as we did already in Section 2.4.

Theorem 3. *Given a variable-free circuit* Γ *such that* $\mathrm{graph}(\Gamma) \subseteq (G \boxtimes H)$ *where* G *is a (rooted) tree and* H *a directed acyclic graph. The evaluation problem for* Γ *can be* AC^1*-reduced to the evaluation problem for a variable-free circuit* Δ *such that* $\mathrm{graph}(\Delta) \subseteq (P \boxtimes H)$ *where* P *is a (directed) path with* $P \subseteq G$.

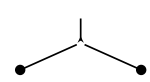
Proof. The initial contraction tree is $\Gamma/\equiv_G$ together with the partial order that G induces on $\Gamma/\equiv_G$. Algorithm 1 requires the contraction tree to be regular. In general $\Gamma/\equiv_G$ is not regular, but we can use a trick to make it regular. We add to each node n of out-degree one a second child node m that has only constant gates such that $n \not\succcurlyeq m$. Thus, the newly added node has no influence at all on the value of the original circuit. Its only function is to make the contraction tree regular and thus guiding the order in which the contraction algorithm contracts the nodes of simple paths in the original tree.

During the contraction process we maintain the following invariant of the contraction tree:

1. The contraction tree is (modulo equivalence) a decomposition $\Gamma/\equiv_F$ of Γ for some graph minor F of G, i.e. $\equiv_F$ is coarser than $\equiv_G$.

2. For each node $N \in \Gamma/\equiv_F$ there is a path $P \subseteq \Gamma/\equiv_G$ in the *initial* tree such that all non-constant gates $g \in N$ are in $\circ_{p \in P}\, p$. Moreover, any node $p \in P$ is a grand-parent of any node $q \in \Gamma/\equiv_G$ with $q \subseteq M$ for $M \in \Gamma/\equiv_F$ being a child of N.

3. A leaf does not contain variable gates.



The first sentence of the invariant simply states that the contraction is sound. Given the soundness, the second sentence implies that all non-constant gates of a node in the contraction tree belong to a subcircuit of eval(Γ) that is a subgraph of normal product of H and a path in G. Actually, the statement is slightly stronger by requiring that the path can be extended to all child nodes in the contraction tree. This is needed in order for the assertion to become inductive. The third sentence guarantees that we can fully evaluate a leaf independently of the rest of the circuit and thus prevent that the parallel contraction gets stuck in a subtree.

It is easy to see that the initial contraction tree fulfills the invariant. Assume we are given an oracle for the evaluation of variable-free circuits that are subgraphs of the normal product of H with a path in G. The contraction step is as follows: Given a node r with child nodes s and t where s is a leaf. As a consequence of the invariant the graph of s is a subgraph of the normal product of a path with H. Since s does not contain variable gates we can use the oracle to evaluate s. The result is a constant circuit. Next we propagate this constants into r in AC^0. Finally we compose r with t. Formally, the contraction step is

$$t \circ (\text{propagate}_s(\text{eval}(s) \circ r)).$$

The evaluation is done using the oracle. The remaining steps can be performed in L. Figure 3.1 illustrates the contraction step and provides an intuition on the role of the invariant.

It remains to show that the contraction step preserves the invariant. The first sentence follows from Lemma 7 and Lemma 6. We know that for r and t all non-constant gates are on a path P_r and P_t, respectively, in the initial tree $\Gamma/\equiv_G$. We further know that all initial-nodes in P_r are grand-parents from all initial nodes in P_t. Hence, we can extend P_r to include P_t in the newly contracted node r'. Since s is a leaf it does not contain any variable gates. After it got evaluated it is constant. Therefore all non-constant nodes in r' are part of P_t or P_r and thus on a path in $\Gamma/\equiv_G$. Moreover, all nodes in the extension of P_r are grand-parents of all grand-children of P_t. We thus established the second sentence of the invariant.

The new node r' is a leaf only if t is a leaf. If t is a leaf it does not contain variable gates. We know that s is a leaf and contains no variables. r does only depend on t and s. Hence r' does not depend on any gate outside of r' and does thus not contain variable gates if it is a leaf. This finishes the proof. □

In the special case that H is a path we prove a stronger reduction.

Theorem 4. *Given a circuit* Γ *with* $\text{graph}(\Gamma) \subseteq (G \boxtimes H)$ *for some (rooted) tree* G *and some (directed) path* H*. The evaluation problem for* Γ *can be* AC^1*-reduced to the evaluation problem for* OIF *circuits with variable gates.*

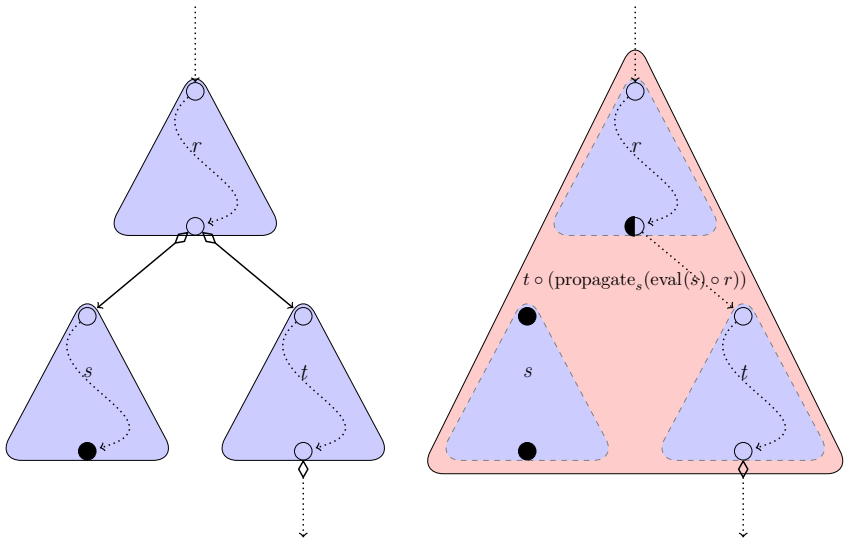
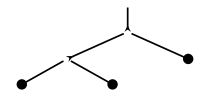


Figure 3.1: Illustration of the contraction step. The figure on the right shows the result of the contraction of the nodes in the figure on the left.
A diamond at an arrow root indicates variable gates. The circles denote subcircuits that are initial nodes (nodes of the initial contraction tree $\Gamma/\equiv G$). A filled circle indicates a constant circuit. A dotted path denotes the path of initial nodes that contains all unevaluated gates.
The node s does not contain variable gates and is constant after evaluation. The constants from s are then propagated into r and the nodes are contracted into a single node.



Proof. The stronger result is obtained from the proof of Theorem 3 through the additional invariant that each node is an evaluated OIF circuit.

Initially, each node is a path. Hence, it is OIF and can be evaluated in parallel using the oracle for OIF circuits. As result we obtain a contraction tree that satisfies the invariant. In order to maintain the stronger invariant we modify the contraction step as follows:

$$\mathrm{eval}_r(t \circ s \circ r)$$

which is the same as

$$(t \setminus \mathrm{const}(t)) \circ \mathrm{eval}(\mathrm{const}(t) \circ s \circ r)$$

First of all observe that the invariant from the proof of Theorem 3 also holds for the new contraction step. The only difference is that we evaluate a contracted node earlier. We do not wait until it becomes a leaf but (partially) evaluate it already when it is created. Figure 3.2 provides an illustration of the modified contraction step.

We have to prove that the contraction step maintains the invariant the all nodes are evaluated and OIF. From the invariant from the proof of Theorem 3 we know that a node of the contraction tree is a subgraph of the normal product of two paths. Hence, all nodes are planar circuits. Initially, all nodes in $\Gamma/\equiv_G$ are OIF. We call the nodes of the initial tree $\Gamma/\equiv_G$ *initial nodes* and we initialize the contraction tree by evaluating each initial node. Using the oracle of the reduction we can do this in parallel in logDCFL. After that the invariant holds for the initial tree.

Assume, we are given a node r with child nodes s and t where s is a leaf such that the invariant holds for r, s, and t. Let $r' = \mathrm{eval}_r(t \circ (s \circ r)$ be the resulting node of the contraction step. Variable gates are always on the outer face of a node, because variables are introduced exclusively by the decomposition and disappear on recomposition. Since s is constant all variable gates of r are the variable gates of t. The nodes s and t are evaluated by assumption. Hence, all constants in s and t are sinks and thus in r' all constants in $t \setminus \mathrm{const}(t)$ are sinks (actually there are no constants at all in $t \setminus \mathrm{const}(t)$). The circuit $\mathrm{const}(t) \circ s \circ r$ is OIF because r is OIF, all edges of r' are edges in r as well. Hence, we can evaluate $\mathrm{const}(t) \circ s \circ r$ using the oracle. As a result all constants in $\mathrm{eval}(\mathrm{const}(t) \circ s \circ r)$ are sink gates. Summing up, we get that all constants in $(t \setminus \mathrm{const}(t) \circ \mathrm{eval}(\mathrm{const}(t) \circ s \circ r)$ are sink gates.

We already know that r' is planar and that all variable gates are the variable gates from t. The induction hypothesis ensures that are on the outer face of t. Since subcircuits are convex and Γ is variable-free (i.e. all variables are introduced by decomposition) it holds that all variable gates of t are on the outer face of r'.

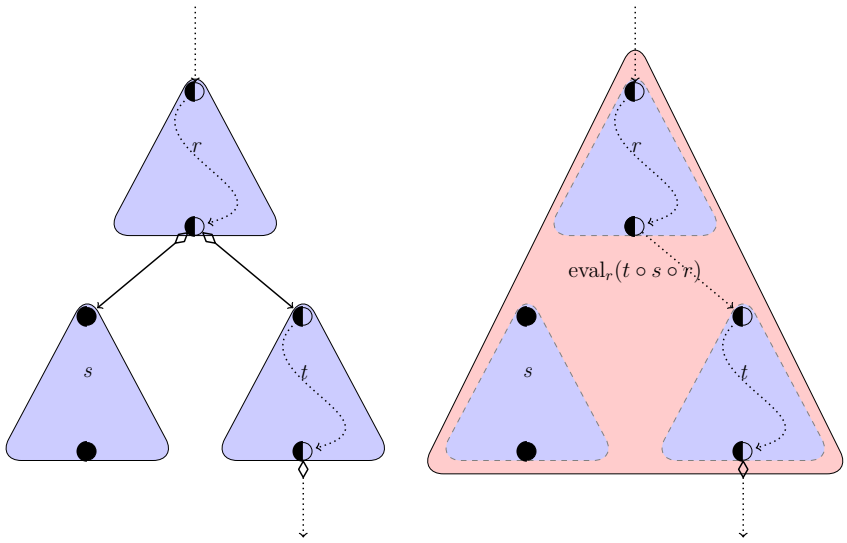
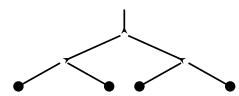


Figure 3.2: Illustration of the contraction step for the reduction to OIF circuits. The figure on the right shows the result of the contraction of the nodes in the figure on the left.
For an explanation of the symbols see Figure 3.1. If the initial node at the root of a contraction node N is filled then N is constant. If it is semi-filled then N is evaluated.
The leaf node s is constant already before the contraction step. The variables gates of r are instantiated with the constant gates from s and t. Then r is evaluated. Since s and t where evaluated already in the beginning the resulting contracted node is evaluated.

We thus can conclude that all nodes are always OIF, which proves the invariant. In the contraction step the evaluation of OIF circuits is done using the reduction oracle. The remaining work in the contraction step can be performed in L. □

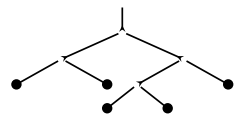


Chapter 4

LTL Model Checking of Restricted Structures

LTL model checking is PSPACE-complete [85]. However, the data complexity is only linear [65] while the expression complexity is PSPACE. The expression complexity of PSPACE for LTL model checking is due the fact that one can encode the computations of a PSPACE Turing machine within an LTL formula [85]. Thus, satisfiability is PSPACE hard. It is easy to see that this holds even for encodings where a superset of all computations of the Turing machine can be represented in a relatively small (polynomial) Kripke structure. Hence, the satisfiability problem can be reduced to the co-model checking problem on this Kripke structure. Therefore, when searching for tractable subproblems of LTL model checking, most research has focused on restrictions of the logic.

Although the high complexity of model checking generally is due to the formula, this does not necessarily imply that this is true also for restricted classes of Kripke structures. The PSPACE hardness results both for the combined as well as for the expression complexity of the problem are worst-case scenarios with respect to the Kripke structure. Thus, it is not obvious if the problem is PSPACE hard except for a rather pathological or degenerated class of structures. E.g. the problem becomes trivial (in fact NC^1-complete [16]) on structures with a singleton state and it is easy to see that the problem is in P on deterministic structures, i.e. on computation paths [71]; actually, it is in NC, as we will prove in this chapter. It thus is reasonable to investigate the complexity of the model checking problem for restricted classes of structures. In this chapter we investigate the complexity of the LTL model checking problem with respect to the Kripke structure. This means that in our results we quantify over all formulas. Moreover, we consider only the frame, i.e. the structure of the underlying graph, of a Kripke structure



and quantify over all possible labelings.

As we will see, the problem is PSPACE hard for a vast class of even very simple Kripke structures. A Kripke structure is called *weak*, if there are no two distinct cycles within a single strongly connected component; in other words: all cycles are pairwise disjoint. This implies that there is a partial ordering with respect to reachability on the cycles of a weak Kripke structure. We will show that the model checking problem is PSPACE hard for any non-weak Kripke structure and that the problem is in coNP for weak Kripke structures. Additionally, we identify some classes of Kripke structures for which the model checking problem can be reduced to checking a polynomial number of finite computation paths. For these classes the model checking problem is in NC. As mentioned before, for all these results we restrict our attention to the frame of a Kripke structure and quantify over all possible labelings.

4.1 Linear-Time Temporal Logic – LTL

We consider linear-time temporal logic (LTL) with the usual finite-path semantics, which includes a weak and a strong version of the Next operator [66]. Let AP be a set of atomic propositions. The *LTL formulas* are defined inductively as follows: every atomic proposition $p \in \mathrm{AP}$ is a formula. If ϕ and ψ are formulas, then so are

$$\neg\phi, \quad \phi \wedge \psi, \quad \phi \vee \psi, \quad \mathrm{X}^{\exists}\,\phi, \quad \mathrm{X}^{\forall}\,\phi, \quad \phi\,\mathrm{U}\,\psi, \quad \text{and} \quad \phi\,\mathrm{R}\,\psi\ .$$

Let $p \in \mathrm{AP}$. We use *true* to abbreviate $p \vee \neg p$ and *false* as an abbreviation for $p \wedge \neg p$. For a formula ϕ we write $\mathrm{G}\,\phi$ to abbreviate $false\,\mathrm{R}\,\phi$ and $\mathrm{F}\,\phi$ as an abbreviation for $true\,\mathrm{U}\,\phi$. The *size of a formula* ϕ is denoted by $|\phi|$.

LTL formulas are evaluated over computation paths over the set of states 2^{AP}. Given an LTL formula ϕ, a nonempty computation path ρ *satisfies* ϕ at position i $(0 \leq i < |\rho|)$, denoted by $(\rho, i) \models \phi$, if one of the following holds:

- $\phi \in \mathrm{AP}$ and $\phi \in \rho_i$,
- $\phi = \neg\psi$ and $(\rho, i) \not\models \psi$,
- $\phi = \phi_l \wedge \phi_r$ and $(\rho, i) \models \phi_l$ and $(\rho, i) \models \phi_r$,
- $\phi = \phi_l \vee \phi_r$ and $(\rho, i) \models \phi_l$ or $(\rho, i) \models \phi_r$,
- $\phi = \mathrm{X}^{\exists}\,\psi$ and $i + 1 < |\rho|$ and $(\rho, i + 1) \models \psi$,
- $\phi = \mathrm{X}^{\forall}\psi$ and $i + 1 = |\rho|$ or $(\rho, i + 1) \models \psi$,
- $\phi = \phi_l\,\mathrm{U}\,\phi_r$ and $\exists i \leq j < |\rho|$ s.t. $(\rho, j) \models \phi_r$ and $\forall i \leq k < j$, $(\rho, k) \models \phi_l$, or

- $\phi = \phi_l \,\mathrm{R}\, \phi_r$ and $\forall i \leq j < |\rho|.(\rho, j) \models \phi_r$ or $\exists i \leq k < j$ s.t. $(\rho, k) \models \phi_l$.

For $|\rho| = \infty$ for any $i \in \mathbb{N}$ it holds that $(\rho, i) \models \mathrm{X}^{\exists}\, \psi$ if and only if $(\rho, i) \models \mathrm{X}^{\forall} \psi$. An LTL formula ϕ is *satisfied* by a nonempty path ρ (denoted by $\rho \models \phi$) iff $(\rho, 0) \models \phi$.

An LTL formula ϕ is said to be in *positive normal form* if in ϕ only atomic propositions appear in the scope of the symbol $\neg$. The following *dualities* ensure that each LTL formula ϕ can be rewritten into a formula ϕ' in positive normal form with $|\phi'| = O(|\phi|)$.

$$\begin{aligned}
&\neg\neg\phi \;\equiv\; \phi \;;\\
&\neg \mathrm{X}^{\forall}\phi \;\equiv\; \mathrm{X}^{\exists}\neg\phi \;;\\
&\neg(\phi_l \wedge \phi_r) \;\equiv\; (\neg\phi_l) \vee (\neg\phi_r) \;;\\
&\neg(\phi_l \,\mathrm{U}\, \phi_r) \;\equiv\; (\neg\phi_l)\,\mathrm{R}(\neg\phi_r) \;.
\end{aligned}$$

The semantics of LTL implies the *expansion laws*, which relate the satisfaction of a temporal formula in some position of the path to the satisfaction of the formula in the next position and the satisfaction of its subformulas in the present position:

$$\begin{aligned}
&\phi_l \,\mathrm{U}\, \phi_r \;\equiv\; \phi_r \vee (\phi_l \wedge \mathrm{X}^{\exists}\,(\phi_l \,\mathrm{U}\, \phi_r)) \;;\\
&\phi_l \,\mathrm{R}\, \phi_r \;\equiv\; \phi_r \wedge (\phi_l \vee \mathrm{X}^{\forall}\,(\phi_l \,\mathrm{R}\, \phi_r)) \;.
\end{aligned}$$

Finally, we provide some notation for speaking conveniently about the syntax of formulas. A formula ψ is a *direct subformula* of a formula ϕ, denoted as $\psi \prec \phi$, if

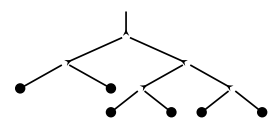
- $\phi = *\psi$ with $* \in \left\{\neg, \mathrm{X}^{\exists}, \mathrm{X}^{\forall}\right\}$, or
- $\phi = \phi_l * \phi_r$ with $* \in \{\wedge, \vee, \mathrm{U}, \mathrm{R}\}$.

The relation $\preceq$ is the reflexive and transitive hull of $\prec$. We say that a formula ψ is a subformula of ϕ if $\psi \preceq \phi$. The set of subformulas $\mathrm{subf}(\phi)$ of ϕ is $\{\psi \mid \psi \preceq \phi\}$. We often identify ϕ with $\mathrm{subf}(\phi)$ and write $\psi \in \phi$ instead of $\phi \in \mathrm{subf}(\phi)$ or $\phi \preceq \psi$. The *formula tree* of a formula ϕ is the rooted tree $\langle \mathrm{subf}(\phi), \prec \rangle$.

Kučera and Strejček prove in [60] a generalized stuttering theorem for LTL that we will use later:

Definition 1 (Generalized Stutter Equivialence)**.** *Given a computation path ρ. A subsequence $\rho_{i,j}$ of ρ is* (m,n)-redundant *if $\rho_{(j+1),(j+1)+m\cdot(j-i)-m+1+n}$ is a prefix of $\rho^{\omega}_{i,j}$.*

We say that two computation paths ρ and σ are (m,n)-stutter equivalent *if ρ is obtained from σ by removing non-overlapping (m,n)-redundant subsequences, or vice versa.*



Theorem 5 (Kučera and Strejček (2005))**.** *Given an LTL formula ϕ with maximal nesting depth of* U *and* R *modalities of m and with maximal nesting depth of* $\mathrm{X}^{\exists}$ *and* $\mathrm{X}^{\forall}$ *modalities of n. The set of $\{\rho \mid \rho \models \phi\}$ is closed under (m,n)-stutter equivalence.*

4.2 Efficient Parallel LTL Path Checking

We now come to the main result of this thesis, namely, that the path checking problem for LTL is in NC, the class of problems that can be solved efficiently in parallel.

Theorem 6. *MC[LTL, path] is in* $\mathsf{AC}^1(\mathsf{logDCFL})$.

The proof proceeds as follows: First the problem is translated into the problem of evaluating a monotone Boolean circuit. Next we use Theorem 4 from Section 3.4 to reduce the problem the problem of evaluating an OIF circuit. Finally we apply Corollary 1 from Section 3.4. Figure 4.1 provides an outline of the proof.

Given an LTL formula ϕ in positive normal from and a finite computation path ρ, the problem of checking ϕ on ρ is translated into the problem of evaluating a circuit. We use the expansion laws of LTL to unfold ϕ over ρ such that each gate of the resulting circuit represents the value of some subformula of ϕ at some position of ρ.

Let $\rho = \rho_0, \ldots, \rho_{n-1}$. We construct the circuit $\mathrm{cir}(\phi, \rho) = \langle \Gamma, \gamma \rangle$ with $\Gamma = \{g_{\psi,\rho_r} \mid \psi \in \phi, 0 \leq r < n\}$ and $\gamma(g_{\psi,\rho_r})$ defined by

- $\rho_r(\psi)$ for $\psi \in \mathrm{AP}$,
- $\neg\rho_r(a)$ for $\psi = \neg a$, $a \in \mathrm{AP}$,
- $\langle \wedge, \langle g_{\chi,\rho_r}, g_{\omega,\rho_r} \rangle \rangle$ for $\psi = \chi \wedge \omega$,
- $\langle \vee, \langle g_{\chi,\rho_r}, g_{\omega,\rho_r} \rangle \rangle$ for $\psi = \chi \vee \omega$,
- $\langle \mathrm{id}, g_{\chi,\rho_{r+1}} \rangle$ for $\psi = \mathrm{X}^{\exists}\chi$ and $r < n-1$,
- 0 for $\psi = \mathrm{X}^{\exists}\chi$ and $r = n-1$,
- $\langle \mathrm{id}, g_{\chi,\rho_{r+1}} \rangle$ for $\psi = \mathrm{X}^{\forall}\chi$ and $r < n-1$,
- 1 for $\psi = \mathrm{X}^{\forall}\chi$ and $r = n-1$,
- $\langle \lambda x, y, z.x \vee (y \wedge z), \langle g_{\omega,\rho_r}, g_{\chi,\rho_r}, g_{\psi,\rho_{r+1}} \rangle \rangle$ for $\psi = \chi \,\mathrm{U}\, \omega$ and $r < n-1$,
- $\langle id, g_{\omega,\rho_r} \rangle$ for $\psi = \chi \,\mathrm{U}\, \omega$ and $r = n-1$,

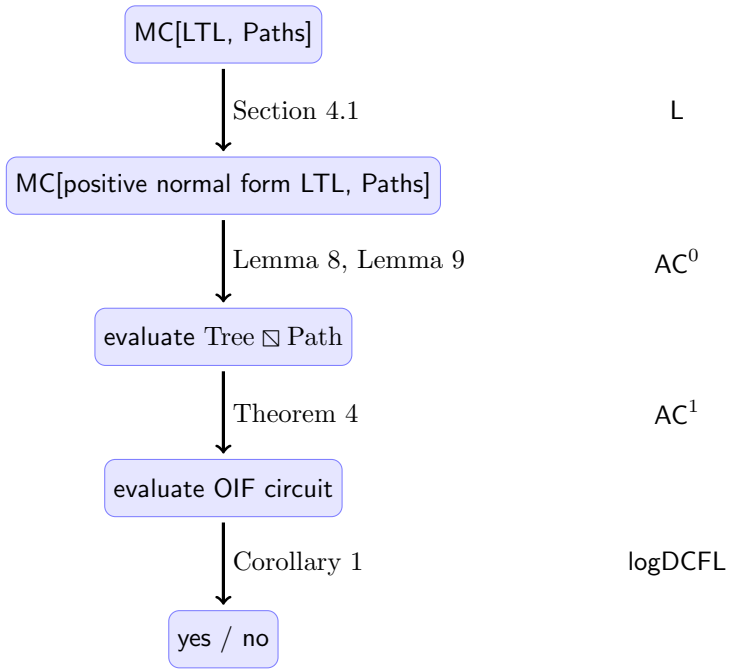
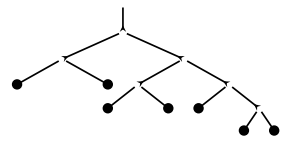


Figure 4.1: Overview over the algorithm for efficient parallel path checking for LTL.



- $\langle \lambda x, y, z.x \wedge (y \vee z), \langle g_{\omega,\rho_r}, g_{\chi,\rho_r}, g_{\psi,\rho_{r+1}} \rangle \rangle$ for $\psi = \chi \,\mathrm{R}\, \omega$ and $r < n-1$, and
- $\langle id, g_{\omega,\rho_r} \rangle$ for $\psi = \chi \,\mathrm{R}\, \omega$ and $r = n-1$,

where $0 \leq r < n$ and $\psi, \chi, \omega \in \phi$. Since the construction of $\mathrm{cir}(\phi, \rho)$ is in AC^0 the following lemma provides us with an AC^0-reduction from the problem of deciding $\rho \models \phi$ to the problem of evaluating $\mathrm{cir}(\phi, \rho)$.

Lemma 8. *Given an LTL formula ϕ in positive normal form and a finite computation path ρ, for the circuit $\langle \Gamma, \gamma' \rangle = \mathrm{eval}(\mathrm{cir}(\phi, \rho))$ it holds that*

$$\gamma'(g_{\psi,\rho_r}) = \begin{cases} 1 & \text{if } \rho, r \models \psi, \\ 0 & \text{otherwise,} \end{cases}$$

for $\psi \in \phi$ and $0 \leq r < n$.

Proof. We prove the lemma by induction over the structure of ϕ. Initially, for the $\psi \in \phi$ being an atomic proposition the statement follows directly from the definition of γ' and the semantics of LTL. Given that the statement holds for each subformula $\chi \in \psi$. If ψ is a disjunction, a conjunction, a $\mathrm{X}^\exists$-formula, or a $\mathrm{X}^\forall$-formula the statement follows directly from the semantics of the LTL and the definition of γ'. For ψ being an U- or a R-formula the statement follows from the definition of γ' and the expansion laws of LTL. □

In order to evaluate the circuit $\mathrm{cir}(\phi, \rho)$ we claim that it is a subgraph of the normal product $\phi \boxtimes \rho$ of the formula tree of ϕ and the path ρ:

Lemma 9. *Given an LTL formula ϕ in positive normal form and a finite computation path ρ, the circuit $\mathrm{cir}(\phi, \rho)$ is a subgraph of $\phi \boxtimes \rho$.*

Proof. Let $e = \langle \langle \phi, \rho_r \rangle, \langle \psi, \rho_s \rangle \rangle$ an edge in $\mathrm{cir}(\phi, \rho)$. From the definition of $\mathrm{cir}(\phi, \rho)$ it follows that either $\phi = \psi \wedge s = r+1$ or $\phi \succ \psi \wedge r = s$ or $\phi \succ \psi \wedge s = r+1$. Hence, e is an edge of $\phi \boxtimes \rho$. □

We are now ready to complete the proof of Theorem 6:

Proof of Theorem 6. Given an LTL formula ϕ and a finite computation path ρ. In L convert ϕ into positive normal form. Use Lemma 8 to AC^0-reduce the problem of deciding $\rho \models \phi$ to the evaluation of $\langle \Gamma, \gamma \rangle = \mathrm{cir}(\phi, \rho)$. By Lemma 9 Γ is a subgraph of the normal product $\phi \boxtimes \rho$. Hence, we can use Theorem 4 to AC^1-reduce the evaluation of Γ to the evaluation OIF circuits with variable gates. Thus, we can use the algorithm from Corollary 1 which runs in $\mathsf{logDCFL}$. The overall complexity is $\mathsf{AC}^1(\mathsf{logDCFL})$. □

4.3 LTL Model Checking Problems in NC

In this section we derive some corollaries from Theorem 6. In general any class of Kripke structures for which the model checking problem can be deterministically reduced to a polynomial number of parallel path checking problems can be model checked in NC. In particular, trees can be decomposed into a linear number of paths:

Corollary 2. *MC[LTL, tree] is in* $\mathsf{AC}^1(\mathsf{logDCFL})$. □

DAGs of constantly bounded depth can be unfolded in trees with only linear blowup:

Corollary 3. *MC[LTL, DAG of depth* $O(1)$*] is in* $\mathsf{AC}^1(\mathsf{logDCFL})$. □

Markey and Schnoebelen present in [71] a reduction from the problem of checking ultimately periodic paths. We provide a more general reduction. We start with an observation about weak Kripke structures:

Lemma 10. *Let* $\mathcal{K}$ *be a weak Kripke structure. Any (finite or infinite) computation path* $\rho \in \mathrm{lang}(\mathcal{K})$ *is of the form* $\left(\prod_{i=0}^{n-1} u_i \cdot v_i^{\alpha_i}\right)$ *with*

- $n \leq |\mathcal{K}|$,
- $\alpha_i \in \mathbb{N}$ *for* $i < n-1$ *and* $\alpha_{n-1} \in \mathbb{N} \cup \{\infty\}$, *and*
- u_i, v_i *are finite paths in* $\mathcal{K}$

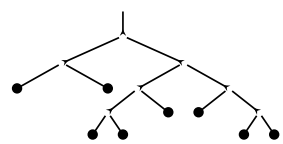
for $0 \leq i < n$.

Proof. The statement of the lemma follows from the fact that for a weak Kripke structure all cycles are disjoint and the cycle-graph is a directed acyclic graph. □

The lemma implies that we can represent a computation path ρ in a weak Kripke structure $\mathcal{K}$ as a path R in the cycle-graph of $\mathcal{K}$ together with the coefficient α_i for each cycle v_i that occurs in ρ. We denote this representation of ρ by $\mathsf{R}_{\alpha_0,\ldots,\alpha_{n-1}}$.

Lemma 11. *Given an LTL formula* ϕ *and a weak Kripke structure* $\mathcal{K}$. *If there is a computation path* $\rho = \mathsf{R}_{\alpha_0,\ldots,\alpha_{n-1}} \in \mathrm{lang}(\mathcal{K})$ *with* $\rho \models \phi$ *then there is a computation path* $\rho' = \mathsf{R}_{\beta_0,\ldots,\beta_{n-1}}$ *with* $\beta_i \leq |\phi|+1$ *such that* $\rho' \models \phi$. *In particular it holds that* $|\rho'| = O(|\phi| \cdot |\mathcal{K}|)$.

Proof. Represent the computation path ρ according to Lemma 10 and apply the generalized stuttering theorem (Theorem 5) from Kučera and Strejček. □



First of all Lemma 11 subsumes the reduction from [71]. By reducing the problem of checking an ultimately periodic path to the finite path checking problem we get the following corollary:

Corollary 4. *MC[LTL, ultimately periodic path] is in ACO*(logDCFL).

Proof. The computation path ρ can be represented as R_∞. Due to Lemma 11 it is sufficient enumerate and check all computation paths R_α for $\alpha \leq |\phi| + 1$. By Theorem 6 each check can be done in $\mathsf{AC}^1(\mathsf{logDCFL})$. Since all checks are independent we can do them all in parallel.

Remark: A more careful interpretation of Theorem 5 would reveal that a single check for $\alpha = |\phi| + 1$ is actually sufficient. □

We can use Lemma 11 to generalize the result to Kripke structures with cycle-graphs of constantly bounded depth.

Corollary 5. *MC[LTL, weak, cycle-graph of depth $O(1)$] is in* $\mathsf{AC}^1(\mathsf{logDCFL})$.

Proof. Unfold the Kripke structure into a tree of linear size and constant depth. Each computation path in the unfolded structure can be represented as $\mathsf{R}_{\alpha_0,\ldots\alpha_n}$ where $n \in \mathbb{N}$ is a constant. Due to Theorem 5 it sufficient to enumerate and check all computation path for $\alpha_i \leq |\phi| + 1$ $(0 \leq i \leq n)$. In total there is a polynomial number of computation paths to be checked. Using Theorem 6 all checks can be done in parallel in $\mathsf{AC}^1(\mathsf{logDCFL})$. □

Let G be the cycle-graph of a Kripke structure $\mathcal{K}$. For a vertex $v \in V(G)$ let

$$\zeta(v) = \alpha(v) + \sum_{\langle w,v\rangle \in E(\mathcal{K})} \beta(v) \cdot \zeta(w)$$

where

$$\alpha(v) = \begin{cases} 1 & \text{if } v \text{ is initial in } \mathcal{K} \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$

$$\beta(v) = \begin{cases} 2 & \text{if } v \text{ represents a cycle of } \mathcal{K} \text{ and} \\ 1 & \text{otherwise, and} \end{cases}$$

the empty sum equals zero. Intuitively, the function $\zeta(v)$ counts the number of paths that lead from an initial state to v, where each cycle occurs either zero or one times in a path. Let $\zeta(\mathcal{K}) = \max_{v \in V(G)} \zeta(v)$.

Corollary 6. *For any class C of weak Kripke structures such that ζ is polynomial in the size of the structure it holds that MC[LTL, C] is in* $\mathsf{AC}^1(\mathsf{logDCFL})$. □

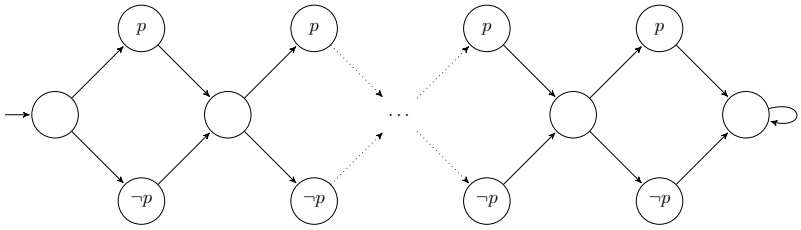


Figure 4.2: Kripke structure used to reduce SAT to LTL model checking

4.4 coNP-Complete LTL Model Checking Problems

In favor of a more concise presentation we exclusively consider LTL over infinite paths throughout the remainder of this chapter.

Theorem 7. *LTL model checking of weak Kripke structures is* coNP*-complete.*

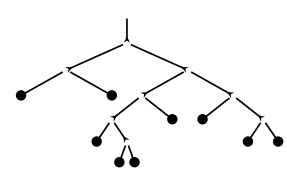
Proof. The proof of the upper bound guesses a possibly infinite path and uses Lemma 11 to reduce the problem to the finite path checking problem. In concrete, given a weak Kripke structure $\mathcal{K}$. In order to decide if $\mathcal{K} \not\models \phi$ guess a path R in the cycle-graph of $\mathcal{K}$ such that there is a path $\rho = \mathsf{R}_{\alpha_0,\ldots,\alpha_{n-1}} \in \mathrm{lang}(\mathcal{K})$ with $\rho \models \neg\phi$. Use Lemma 11 to reduce this to checking $\rho' \models \neg\phi$ for a finite path ρ' of polynomial length. Do this check by use of Theorem 6 in $\mathsf{AC}^1(\mathsf{logDCFL}) \subseteq \mathsf{P}$. Hence the model checking problem for weak Kripke structures is in coNP.

The proof of the lower bound reduces the satisfiability problem of propositional logic to the model checking problem of weak Kripke structures. The reduction is very similar to the reduction used by [85] to show that the co-model checking problem for the fragment of LTL that has F as the only modality is NP-hard.

Given a propositional logic formula f over the set of variables $\{v_0, \ldots, v_n\}$, $n \in \mathbb{N}$. We obtain the LTL formula ϕ from f by substituting for all $0 \leq i \leq n$ each occurrence of the variable v_i by the LTL formula $\mathrm{X}^{\exists^{2i+1}} p$, where $p \in \mathrm{AP}$. It is easy to see that f is satisfiable if and only if ϕ holds on the Kripke structure $\mathcal{K}$ shown in Figure 4.2. □

The above proof actually provides a slightly stronger result:

Corollary 7. *The problem of model checking LTL on planar acyclic graphs is* coNP*-hard.*



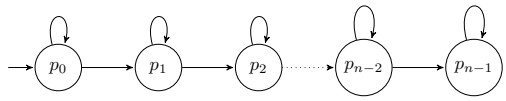


Figure 4.3: Kripke structure used to reduce SAT to LTL model checking of Kripke structures for which the cycle-graph is a path.

There are more classes of Kripke structures with a coNP lower bound. In order to prepare the proof of the next theorem we show here the construction for Kripke structures with a cycle-graph that is a path. In fact self loops, i.e. state-stuttering is sufficient. The idea is similar to the lower bound from the previous theorem but the diamond shaped substructures are replaced by self loops. For a propositional formula f with variables $v_0, \ldots, v_{n-1}$ we build a Kripke structure $\mathcal{K}$ that is a sequence of n self loops as shown in Figure 4.3 where each vertex is labeled with a unique state (represented as a propositional formula) p_i. The LTL formula ϕ is obtained by substituting in f each variable v_i with the LTL formula $\mathrm{F}(p_i \wedge \mathrm{X}^{\exists} p_i)$. It is easy to check that f is satisfiable if and only if $\mathcal{K} \not\models \neg\phi$.

The next theorem is a refined (though more technical) version of Theorem 7. Recall that the function ζ from Section 4.3 counts the number of paths in a weak Kripke structure where each cycle occurs at most once.

Theorem 8. *For any class C of weak Kripke structures for which ζ is exponential in the size of the structure it holds that MC[LTL, C] is complete for* coNP.

The upper bound remains the same since for each weak Kripke structure ζ is exponentially bounded. The lower bound is refined through a stronger constraint on the classes of structures.

Proof. The proof for the lower bound combines the proof for the lower bound for planar DAGs and the lower bound for paths Kripke structures with a cycle-graph that is a path. Again, we reduce SAT to the co-model checking problem on C. Let f a propositional formula with variables $v_0, \ldots, v_{n-1}$. There is a Kripke structure $\mathcal{K}$ in C such that the cycle-graph of $\mathcal{K}$ contains sequence of vertices $v_0, \ldots, v_{n-1}$, where v_i is reachable from v_{i-1} and $\zeta(v_i) = O(2 \cdot v_{i+1})$ (the only possibility to reach an exponential growth is via duplicating the value from a vertex). Moreover, we know that either there are two distinct paths from v_{i-1} to v_i with two distinguishing vertices v_i^0 and v_i^1, or v_i represents a cycle. In the former case we label v_i^0 with a unique label p_i (represented by a propositional formula) and substitute v_i in with the LTL formula $\mathrm{F}\, p_i$. In the latter case we label v_i with a unique label p_i (represented as a propositional formula) and substitute any occurrence of v_i with the LTL formula $\mathrm{F}(p_i \wedge \mathrm{X}^{\exists}(\mathit{true}\,\mathrm{U}\, p_i))$. We

call the resulting LTL formula ϕ. Again, it is easy to check that $\mathcal{K} \not\models \neg\phi$ if and only if f is satisfiable. □

The classification of Kripke structures between NC and coNP-hardness is not a complete dichotomy but there is a gap concerning the structures with $n^{O(1)} \ll \zeta(k) \ll O(2^n)$. This is illustrated via the following corollary.

Corollary 8. *For any class C of weak Kripke structures with $\zeta = O(n^{\log^{O(1)} n})$, where n is the size of a Kripke structure, it holds that MC[LTL, C] is in* polyL*.*

Proof. We can enumerate all computation paths of polynomial length that are relevant according to Lemma 11 in polyL. Each computation path can be checked in $\mathsf{AC}^1(\mathsf{logDCFL}) \subseteq \mathsf{polyL}$. □

4.5 PSPACE-Complete LTL Model Checking Problems

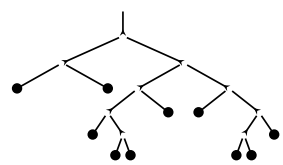
We conclude this chapter by investigating how complex a Kripke structure has to be in order for the model checking problem to become PSPACE-hard. As it turns out the LTL model checking is PSPACE-complete for *any* non-weak Kripke structure. In contrast to the previous results, we get a lower bound that does not depend on the asymptotic behavior of a class of Kripke structures but holds for each structure that is non-weak. Moreover, together with Theorem 7 we get a dichotomic classification.

Theorem 9. *The LTL model checking problem is* PSPACE*-complete for any non-weak Kripke structure.*

Proof. Given a non-weak Kripke structure $\mathcal{K}$. We reduce the validity problem for LTL to the co-model checking problem on $\mathcal{K}$.

We start by choosing an adequate labeling for $\mathcal{K}$. Let $s, t, u \in 2^{\mathrm{AP}}$ be pairwise disjoint states represented as Boolean formulas. Because $\mathcal{K}$ is non-weak we know that there are two distinct cycles that share a common vertex x. Label x with state s. There is a vertex y that is present in only one of the two cycles. Label y with state t. Label all remaining vertices of $\mathcal{K}$ with u. Figure 4.4 provides a schematic view of $\mathcal{K}$.

Given some formula ζ with only a single variable. Deciding validity for a LTL formulas with only a single variable is PSPACE-hard [23]. ζ is valid if and only if $\neg\zeta$ is not satisfiable. To decide if $\phi = \neg\zeta$ is satisfiable is the co-model checking problem on a universal Kripke structure. We will reduce the latter for ϕ to the co-model checking problem on the Kripke structure $\mathcal{K}$ with the labeling given above for a formula ϕ^* that can be constructed from ϕ in L. Since PSPACE



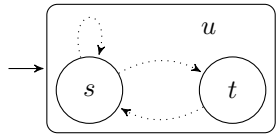


Figure 4.4: Non-weak Kripke structure with the labeling used in the proof of Theorem 9

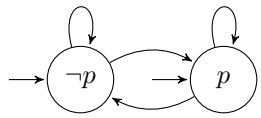


Figure 4.5: The Kripke structure that represents the universal language $\{p, \neg p\}^\omega$.

is closed under complement, the model checking problem for $\mathcal{K}$ is thus PSPACE hard.

We assume that the unique atomic proposition that occurs in ϕ is p. A Kripke structure with two states, namely p and $\neg p$, that represents the universal language $\{p, \neg p\}^\omega$ is shown in Figure 4.5.

In the following we call a suffix of a computation path an a-suffix if the first state of the suffix is a. We call a cycle in a Kripke structure an a-cycle if it starts in an a-state. We identify the cycle with the corresponding state sequence.

The construction of ϕ^* is as follows: First, transform ϕ into positive normal form in L. Next, we define inductively a formula ϕ'. For the cases that ϕ is either an atomic proposition or a negated atomic proposition let

- $p' = s \wedge \mathrm{X}^\exists\,(u\,\mathrm{U}\,s)$ and
- $(\neg p') = s \wedge \mathrm{X}^\exists\,\big(u\,\mathrm{U}\,\big(t \wedge \mathrm{X}^\exists\,(u\,\mathrm{U}\,s)\big)\big)$.

The idea is that the formula p' holds exclusively on an s-cycle that does not include the t state, whereas $(\neg p)'$ holds on any s-cycle that visits the t state. This way, each s-cycle encodes a state of the original Kripke structure. The formula will translate each single step in the original Kripke structure into an s-cycle in $\mathcal{K}$. The remaining cases for ϕ' are defined inductively as follows:

- $\psi' \wedge \chi'$ for $\phi = \psi \wedge \chi$,
- $\psi' \vee \chi'$ for $\phi = \psi \vee \chi$,
- $s \wedge \mathrm{X}^\exists\,(\neg s\,\mathrm{U}\,\psi')$ for $\phi = \mathrm{X}^\exists\,\psi$,

- $s \wedge ((s \rightarrow \psi') \,\mathrm{U}\, \chi')$ for $\phi = \psi \,\mathrm{U}\, \chi$, and
- $s \wedge (\psi' \,\mathrm{R}\, (s \rightarrow \chi'))$ for $\phi = \psi \,\mathrm{R}\, \chi$.

For any formula ϕ we want that a computation path that models ϕ' is "meaningful" in the sense that it can be mapped to a computation path having only p and $\neg p$ states. Therefore we defined $'$ such that a formula ϕ' holds on a computation path ρ only if ρ starts in the s state.

Finally, we set $\phi^* = (\mathrm{G}\,\mathrm{F}\, s) \wedge (\neg s \,\mathrm{U}\, \phi')$. By requiring that any computation path ρ with $\rho \models \phi^*$ returns to s infinitely often, we prevent that ρ "gets stalled" in some "meaningless" loop. The formula allows the computation to reach s initially, and then it forces the remaining computation path to satisfy ϕ' which encodes on $\mathcal{K}$ the original meaning of ϕ.

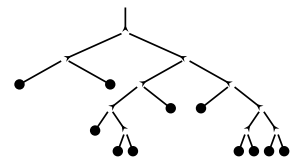
We claim that for each computation path ρ it holds that $\rho \models \phi$ if and only if there is a computation path $\rho^* \in \mathrm{lang}(\mathcal{K})$ such that $\rho^* \models \phi^*$. Let c be a s-cycle in $\mathcal{K}$ that does not visit t and let d an s-cycle in $\mathcal{K}$ that visits t. For proving the "only if" part of the claim assume that $\rho \models \phi$. Let ρ' be defined as follows:

$$\rho' = \begin{cases} c \cdot \rho'_{1,\ldots} & \text{if } \rho_0(p), \text{ and} \\ d \cdot \rho'_{1,\ldots} & \text{otherwise.} \end{cases}$$

It holds that $\rho' \in \mathrm{lang}(\mathcal{K})$, every suffix of ρ' contains an s-state, and ρ' starts with s. Further $'$ induces a surjective mapping from the suffixes of ρ to s-suffixes of ρ'. This mapping is monotonic with respect to the order of start position of the suffixes.

Let e be the (possibly empty) sequence of states on a shortest path in $\mathcal{K}$ that leads from an initial state to s. Let $\rho^* = e \cdot \rho'$. Every suffix of ρ^* contains an s state and therefore it holds that $\rho^* \models \mathrm{G}\,\mathrm{F}\, s$. Since all states in e contradict s from the definition of ρ^* it follows directly that $\rho^* \models (\neg s) \,\mathrm{U}\, \phi'$ if $\rho' \models \phi'$. We prove this by induction over ϕ:

- For $\phi = p$ it holds that if ρ starts with p then ρ' starts with c followed by an s state. Hence $\rho' \models s \wedge \mathrm{X}^{\exists}\, (u \,\mathrm{U}\, s)$.
- For $\phi = \neg p$ it holds that if ρ starts with $\neg p$ then ρ' starts with d followed by an s state. Hence $\rho' \models s \wedge \mathrm{X}^{\exists}\, \big(u \,\mathrm{U}\, \big(t \wedge \mathrm{X}^{\exists}\, (u \,\mathrm{U}\, s)\big)\big)$.
- For $\phi \in \{\psi \wedge \chi, \psi \vee \chi\}$ the claim follows directly from the induction hypothesis and the semantics of LTL.
- For $\phi = \mathrm{X}^{\exists}\, \psi$ it holds that $\rho_{1,\ldots} \models \psi$. By induction hypothesis $(\rho_{1,\ldots})' \models \psi'$ and by definition of ρ' it holds that $\rho' \models s \wedge \mathrm{X}^{\exists}\, (\neg s \,\mathrm{U}\, \psi')$.



- For $\phi = \psi \,\mathrm{U}\, \chi$ there is a position i such that $\rho_{i,\dots} \models \chi$ and $\rho_{j,\dots} \models \phi$ for all $j < i$. By induction hypothesis there is an l such that $\rho'_{l,\dots} \models \chi'$. Since $'$ is surjective and monotonic on the s-suffixes of ρ' from the induction hypothesis it follows that $\rho'_{j,\dots} \models \psi'$ for all s-suffixes with $j < l$. Further recall that ψ' holds only on computation paths that start with s. Hence, for all non-s-suffix $\rho'_{j,\dots}$ with $j < l$ the formula $(s \rightarrow \psi')$ holds trivially. Thus, for all $j < l$ it holds that $\rho'_{j,\dots} \models (s \rightarrow \psi')$ and we get $\rho' \models s \wedge (s \rightarrow \psi') \,\mathrm{U}\, \chi'$.

- The case for $\phi = \psi \,\mathrm{R}\, \chi$ is analogous to the previous case.

We now prove the "if" part of the claim. Given a computation path in $\sigma \in \mathrm{lang}(\mathcal{K})$ such that $\sigma \models \phi^*$. There is a position i_0 such that for all $j < i_0$ it holds that $\sigma_{j,\dots} \models \neg s$ and $\sigma_{i_0,\dots} \models \phi'$. Let $\sigma^0 = \sigma_{i_0,\dots}$. We show that $\sigma^0 \models \phi'$ implies that there is a computation path σ' with $\sigma' \models \phi$. We know that σ^0 contains only s, t, and u states. Moreover, we know from the definition of ϕ^* that any suffix of σ^0 contains an s state. The computation path σ' is defined as follows:

$$\sigma' = \begin{cases} p \cdot \sigma'_{s_0,\dots} & \text{if } \sigma^0_{0,s_0} \text{ contains no } t \text{ state, and} \\ \neg p \cdot \sigma'_{s_0,\dots} & \text{otherwise,} \end{cases}$$

where s_0 is the position of the first s-state in $\sigma_{1,\dots}$. Note that $'$ induces a monotonic and surjective mapping from the s-suffixes of σ^0 to the suffixes of σ. We show by induction over ϕ that $\sigma' \models \phi$:

- For $\phi = p$ we have $\sigma^0 \models s \wedge \mathrm{X}^{\exists}(u \,\mathrm{U}\, s)$. This implies that σ^0_{0,s_0} does not contain any t state and therefore $\sigma'_0(p)$.

- For $\phi = \neg p$ we have $\sigma^0 \models s \wedge \mathrm{X}^{\exists}\big(u \,\mathrm{U}\, \big(t \wedge \mathrm{X}^{\exists}(u \,\mathrm{U}\, s)\big)\big)$. Therefore σ^0_{0,s_0} contains a t state and hence $\sigma'_0(\neg p)$.

- For $\phi \in \{\psi \wedge \chi, \psi \vee \chi\}$ the claim follows directly from the induction hypothesis and the semantics of LTL.

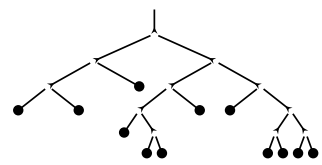
- For $\phi = \mathrm{X}^{\exists}\,\psi$ we have $\sigma^0 \models s \wedge \mathrm{X}^{\exists}(\neg s \,\mathrm{U}\, \psi')$. This implies that $\sigma^0_{s_0,\dots} \models \psi'$. From the induction hypothesis it follows that $\sigma'_{1,\dots} \models \psi$ and thus $\sigma' \models \mathrm{X}^{\exists}\,\psi$.

- For $\phi = \psi \,\mathrm{U}\, \chi$ we have that $\sigma^0 \models s \wedge ((s \rightarrow \psi') \,\mathrm{U}\, \chi')$. Therefore there is an $i \in \mathbb{N}$ such that $\sigma^0_{i,\dots} \models \chi'$ and for all $j < i$ it holds that $\sigma^0_{j,\dots} \models (s \rightarrow \psi')$. By induction hypothesis there is an $l \in \mathbb{N}$ such that $\sigma'_{l,\dots} \models \chi$. For all s-suffixes $\sigma^0_{j,\dots}$ with $j < i$ it holds that $\sigma^0 j, \dots \models \psi'$. Recall that $'$ induces a monotonic and surjective function from the s-suffixes of σ^0 to the suffixes of σ'.

  Together with the induction hypothesis we deduce that for all $j < l$ it holds that $\sigma'_{j,\dots} \models \psi$. We conclude that $\sigma' \models \psi \,\mathrm{U}\, \chi$.

- The case for $\phi = \psi \,\mathrm{R}\, \chi$ is analogous to the previous case. Note, however, that there are infinitely many s-states in σ^0.

□



Chapter 5

Computation Tree Logic Tree Checking

The model checking problem for CTL is tractable, namely it is P-complete. In [18] Clarke, Emerson, and Sistla provide an algorithm with bilinear running time. In [13] Bernholtz, Vardi, and Wolper improve over this and show that CTL model checking is in space linear in the formula and poly-logarithmic in the size of the Kripke structure. Although in the community this bound was considered folklore, the first published proof of the P lower bound that we know of is by Schnoebelen in [84]. It reduces the monotone circuit value problem to MC[CTL, DAGs]. The proof uses $EX^{\exists}$ and $AX^{\exists}$ as only temporal operators. The argument can be easily adapted to use only EF and AF or EU or AU, respectively. There is a long line of research about the complexity of model checking for all kinds of extensions and restriction of CTL (cf. [14] for further references). In [14] the authors comprehensively investigate the complexity of the model checking problem for fragments of the logic.

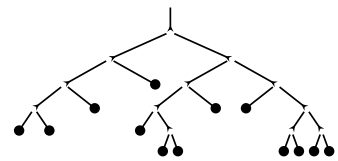
We are not aware of any systematic study of the complexity of the model checking problem for CTL on restricted classes of structures. In the following we summarize some obvious facts. CTL and LTL coincide on computation paths. Hence we get the following corollary.

Corollary 9. *MC[CTL, path] is in* $\mathsf{AC}^1(\mathsf{logDCFL})$.

The next corollary is an immediate consequence from the proof of the P upper bound of CTL model checking from [84].

Corollary 10. *MC[CTL, DAG] is* P*-complete.*

For the last corollary we use the same reduction from [84] as for the previous result. However, this time we start the reduction from the monotone and planar



circuit value problem. In [17] the authors notice that the monotone planar circuit value problem is L hard. As a result we conclude that

Corollary 11. *MC[CTL, planar DAG] is* L*-hard.*

However, the hardness depends on the encoding of the input. Here we assume that the input structure is given as an unsorted set of gates. It is an open problem if this lower bound still holds if input DAG is given along with a topological sorting? To the best of our knowledge the best known upper bound for MC[CTL, planar DAGs] is P. Hence, the exact complexity of model checking of CTL over planar DAGs is open.

The remainder of this chapter is dedicated to our main result on CTL model checking, namely that CTL tree checking is in NC. We start in the next section with the formal definition of the logic CTL.

5.1 Computation Tree Logic – CTL

Analogously to LTL also for CTL we define a weak and a strong version of the *Next*-Operators. Again, let AP be a set of atomic propositions. CTL distinguishes between *path formulas* and *state formulas*. Each atomic proposition $p \in \mathrm{AP}$ p is state formula. If ϕ and χ are state formulas and ψ is a path formula then the following are state formulas:

$$\neg\phi, \quad \phi \wedge \chi, \quad \phi \vee \chi, \quad \mathrm{E}\,\psi, \text{ and } \mathrm{A}\,\psi \ .$$

If ϕ and χ are state formula then the following are path formulas:

$$\mathrm{X}^{\exists}\,\phi, \quad \mathrm{X}^{\forall}\phi, \quad \phi\,\mathrm{U}\,\chi, \text{ and } \phi\,\mathrm{R}\,\chi \ .$$

The CTL formulas are the set of all state formulas. Usually we write $\phi\,\mathrm{EU}\,\chi$, $\phi\,\mathrm{AU}\,\chi$, $\phi\,\mathrm{ER}\,\chi$, and $\phi\,\mathrm{AR}\,\chi$ instead of $\mathrm{E}\,(\phi\,\mathrm{U}\,\chi)$, $\mathrm{A}\,(\phi\,\mathrm{U}\,\chi)$, $\mathrm{E}\,(\phi\,\mathrm{R}\,\chi)$, and $\mathrm{A}\,(\phi\,\mathrm{R}\,\chi)$, respectively.

CTL formulas are evaluated over computation trees. Given a CTL formula ϕ, a nonempty computation tree $\tau \neq \epsilon$ *satisfies* ϕ, denoted by $\tau \models \phi$, if one of the following holds:

- $\phi \in \mathrm{AP}$ and $\phi \in \mathrm{root}(\tau)$,
- $\phi = \neg\psi$ and $\tau \not\models \psi$,
- $\phi = \chi \wedge \psi$ and $\tau \models \chi$ and $\tau \models \psi$,
- $\phi = \chi \vee \psi$ and $\tau \models \chi$ or $\tau \models \psi$,

- $\phi = \mathrm{E}\,\xi$ and there is a rooted computation path ρ in τ such that $\rho \models \xi$, or
- $\phi = \mathrm{A}\,\xi$ and for each rooted computation path ρ in τ it holds that $\rho \models \xi$,

where ψ and χ are state formulas and ξ is a path formula and for a path ρ it holds that $\rho \models \xi$ if one of the following holds:

- $\phi = \mathrm{X}^{\exists}\,\psi$ and $|\rho| > 1$ and $\rho_{1,\ldots} \models \psi$,
- $\phi = \mathrm{X}^{\forall}\psi$ and either $|\rho| = 1$ or $\rho \models \mathrm{X}^{\exists}\,\psi$,
- $\phi = \chi\,\mathrm{U}\,\psi$ and $\exists 0 \leq i < |\rho|.\rho_{i,\ldots} \models \psi \wedge (\forall 0 \leq j < i.\rho_{j,\ldots} \models \chi)$, or
- $\phi = \chi\,\mathrm{R}\,\psi$ and $\forall 0 \leq i < |\rho|.\rho_{i,\ldots} \models \psi \vee (\exists 0 \leq k < j.\rho_{j,\ldots} \models \chi)$.

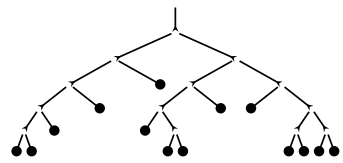
An CTL formula ϕ is said to be in *positive normal form* if in ϕ only atomic propositions appear in the scope of the symbol $\neg$. The following *dualities* ensure that each CTL formula can be rewritten into positive normal form with only linear size:

$$
\begin{aligned}
&\neg\neg\phi \equiv \phi \ ;\\
&\neg\,\mathrm{AX}^{\forall}\,\phi \equiv \mathrm{EX}^{\exists}\,\neg\phi \ ;\\
&\neg\,\mathrm{EX}^{\forall}\,\phi \equiv \mathrm{AX}^{\exists}\,\neg\phi \ ;\\
&\neg(\phi_l \wedge \phi_r) \equiv (\neg\phi_l) \vee (\neg\phi_r) \ ;\\
&\neg(\phi_l\,\mathrm{EU}\,\phi_r) \equiv (\neg\phi_l)\,\mathrm{AR}(\neg\phi_r) \ ;\\
&\neg(\phi_l\,\mathrm{AU}\,\phi_r) \equiv (\neg\phi_l)\,\mathrm{ER}(\neg\phi_r) \ .
\end{aligned}
$$

The semantics of CTL implies the *expansion laws*, which relate the satisfaction of a temporal formula in some position of the path to the satisfaction of the formula in the next position and the satisfaction of its subformulas in the present position:

$$
\begin{aligned}
\phi_l\,\mathrm{EU}\,\phi_r &\equiv \phi_r \vee (\phi_l \wedge \mathrm{EX}^{\exists}\,(\phi_l\,\mathrm{EU}\,\phi_r)) \ ;\\
\phi_l\,\mathrm{AU}\,\phi_r &\equiv \phi_r \vee (\phi_l \wedge \mathrm{AX}^{\exists}\,(\phi_l\,\mathrm{AU}\,\phi_r)) \ ;\\
\phi_l\,\mathrm{ER}\,\phi_r &\equiv \phi_r \wedge (\phi_l \vee \mathrm{EX}^{\forall}\,(\phi_l\,\mathrm{ER}\,\phi_r)) \ ;\\
\phi_l\,\mathrm{AR}\,\phi_r &\equiv \phi_r \wedge (\phi_l \vee \mathrm{AX}^{\forall}\,(\phi_l\,\mathrm{AR}\,\phi_r)) \ .
\end{aligned}
$$

For a CTL formula ϕ the set of subformulas $\mathrm{subf}(\phi)$ includes only state formulas, i.e. proper CTL formulas. Otherwise the definition is analogous to the definition of subformulas of an LTL formula in Section 4.1.



5.2 Efficient Parallel CTL Tree Checking

In this section we prove that tree checking for CTL is efficiently parallelizable.

Theorem 10. *Let $\mathcal{T}$ be the class of Kripke structures that are (finite) trees.*

MC[CTL, $\mathcal{T}$] is in $\mathsf{AC}^2(\mathsf{logDCFL})$.

The idea of the proof is analogous to the result on LTL path checking. There we used parallel tree contraction to evaluate the formula tree over the finite computation path. In the case of CTL tree checking also the Kripke structure over which the formula is evaluated is a tree. Fortunately, these two trees are combined orthogonally by the expansion laws for CTL. The expansion laws allow us to unroll the formula over the finite computation tree such that the resulting combinatorial circuit is a subgraph of the normal product of the formula tree and the computation tree. Thus, we can use the reduction techniques from Section 3.4. In a first step the computation tree is contracted which results in an AC^1-reduction to the evaluation of a circuit that is a subgraph of the normal product of a path and the formula tree. In a second step the formula tree is contracted which results in a reduction to the evaluation of a circuit that is a subgraph of the normal product of two paths; in fact the circuit is OIF. There is one sole problem with this approach: The structure tree is in general not binary. We solve this by providing a L-reduction from general tree structures to binary tree structures. This reduction is applied before the model checking problem is translated into a circuit evaluation problem. An overview of the construction is shown on Figure 5.1. All together we thus get an algorithm that consists of two initial many-one reductions followed by a simple transformation and two nested parallel tree contractions with OIF circuit evaluation as the base case of the reduction chain. The overall complexity therefore is $\mathsf{AC}^0(\mathsf{AC}^1(\mathsf{AC}^1(\mathsf{logDCFL}))) = \mathsf{AC}^2(\mathsf{logDCFL}) \subseteq \mathsf{SAC}^3$.

In concrete we start by L-reducing the problem MC[CTL, $\mathcal{T}$] for general finite trees to the problem MC[CTL, $\mathcal{T}_2$], where $\mathcal{T}_2$ is the class of Kripke structures that are finite binary trees. Given a CTL formula ϕ and a finite computation tree $\tau \in \mathcal{T}$, replace each node n in τ of degree $d(n) > 2$ with a regular binary tree $\nu(n)$ such that the leaves of $\nu(n)$ are the children of n and $\mathrm{root}(\nu(n)) = n$. Call the resulting tree $\nu(\tau)$. Note that the size of $\nu(\tau)$ is at most twice the size of τ. Label the new nodes $V(\nu(\tau)) \setminus V(\tau)$ with a fresh (non-empty) state s such that $s \cap s' = \emptyset$ for all labels s' occurring in τ. We define the CTL formula ϕ' inductively as follows:

- p for $\phi = p \in \mathrm{AP}$,
- $\neg\chi'$ for $\phi = \neg\chi$,
- $\chi' \wedge \psi'$ for $\phi = \chi \wedge \psi$,

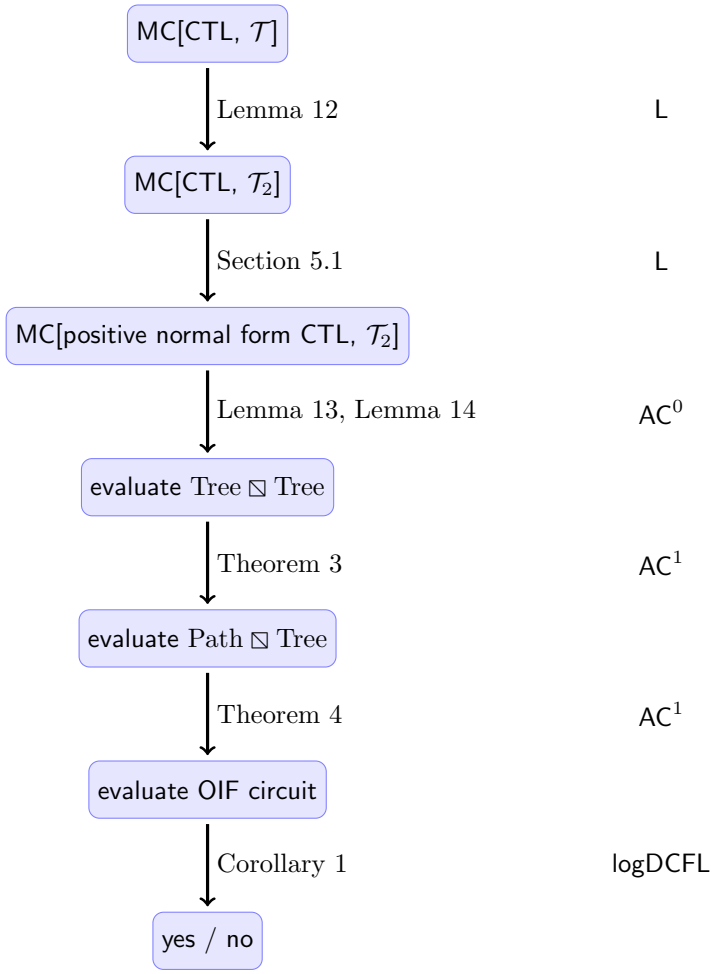
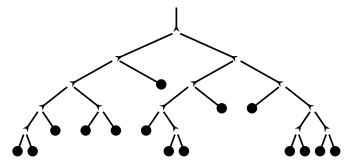


Figure 5.1: Overview over the algorithm for efficient tree checking for CTL.



- $\chi' \vee \psi'$ for $\phi = \chi \vee \psi$,
- $\mathrm{EX}^{\exists}(s\,\mathrm{EU}(\neg s \wedge \chi'))$ for $\phi = \mathrm{EX}^{\exists}\chi$,
- $\mathrm{AX}^{\exists}(s\,\mathrm{AU}(\neg s \wedge \chi'))$ for $\phi = \mathrm{AX}^{\exists}\chi$,
- $\mathrm{EX}^{\forall}(s\,\mathrm{EU}(\neg s \wedge \chi'))$ for $\phi = \mathrm{EX}^{\forall}\chi$,
- $\mathrm{AX}^{\forall}(s\,\mathrm{AU}(\neg s \wedge \chi'))$ for $\phi = \mathrm{AX}^{\forall}\chi$,
- $(s \vee \chi')\,\mathrm{EU}\,(\neg s \wedge \psi')$ for $\phi = \chi\,\mathrm{EU}\,\psi$,
- $(s \vee \chi')\,\mathrm{AU}\,(\neg s \wedge \psi')$ for $\phi = \chi\,\mathrm{AU}\,\psi$,
- $(\neg s \wedge \chi')\,\mathrm{ER}\,(s \vee \psi')$ for $\phi = \chi\,\mathrm{ER}\,\psi$, and
- $(\neg s \wedge \chi')\,\mathrm{AR}\,(s \vee \psi')$ for $\phi = \chi\,\mathrm{AR}\,\psi$.

Lemma 12. *Given a CTL formula ϕ and a tree $\tau \in \mathcal{T}$.*

$$\tau \models \phi \text{ iff } \nu(\tau) \models \phi'.$$

$\nu(\tau)$ and ϕ' can be constructed in L.

Proof. By induction over ϕ. □

Given a CTL formula ϕ in positive normal from and a finite computation tree τ, using Lemma 12 we can assume that τ is binary. The problem of checking ϕ on τ is translated into the problem of evaluating a circuit. We use the expansion laws of CTL to unfold ϕ over τ such that each gate of the resulting circuit represents the value of some subformula of ϕ at some node in τ. We construct the circuit $\mathrm{cir}(\phi, \tau) = \langle \Gamma, \gamma \rangle$ with $\Gamma = \{g_{\psi,t} \mid \psi \in \mathrm{subf}(\phi), t \in \tau\}$ and $\gamma(g_{\psi,t})$ defined by

- $r(\psi)$ for $\psi \in \mathrm{AP}$,
- $\neg r(a)$ for $\phi = \neg a$, $a \in \mathrm{AP}$,
- $\langle \wedge, \langle g_{\chi,r}, g_{\omega,r} \rangle \rangle$ for $\psi = \chi \wedge \omega$,
- $\langle \vee, \langle g_{\chi,r}, g_{\omega,r} \rangle \rangle$ for $\psi = \chi \vee \omega$,
- $\langle \wedge, \langle g_{\chi,s}, g_{\chi,t} \rangle \rangle$ for $\psi = \mathrm{AX}^{\exists}\chi$ or $\psi = \mathrm{AX}^{\forall}\chi$ and r not a leaf,
- $\langle \vee, \langle g_{\chi,s}, g_{\chi,t} \rangle \rangle$ for $\psi = \mathrm{EX}^{\exists}\chi$ or $\psi = \mathrm{EX}^{\forall}\chi$ and r not a leaf,
- 0 for $\psi = \mathrm{EX}^{\exists}\chi$ or $\psi = \mathrm{AX}^{\exists}\chi$ and r a leaf,
- 1 for $\psi = \mathrm{EX}^{\forall}\chi$ or $\psi = \mathrm{AX}^{\forall}\chi$ and r a leaf,

- $\langle \lambda w, x, y, z.w \vee (x \wedge (y \vee z)), \langle g_{\omega,r}, g_{\chi,r}, g_{\psi,s}, g_{\psi,t} \rangle \rangle$ for $\psi = \chi \,\mathrm{EU}\, \omega$ and r not a leaf,
- $\langle \lambda w, x, y, z.w \vee (x \wedge (y \wedge z)), \langle g_{\omega,r}, g_{\chi,r}, g_{\psi,s}, g_{\psi,t} \rangle \rangle$ for $\psi = \chi \,\mathrm{AU}\, \omega$ and r not a leaf,
- $\langle id, g_{\omega,r} \rangle$ for $\psi = \chi \,\mathrm{EU}\, \omega$ or $\psi = \mathrm{AU}\, \omega$ and r leaf,
- $\langle \lambda w, x, y, z.w \wedge (x \vee (y \vee z)), \langle g_{\omega,r}, g_{\chi,r}, g_{\psi,s}, g_{\psi,t} \rangle \rangle$ for $\psi = \chi \,\mathrm{ER}\, \omega$ and r not a leaf,
- $\langle \lambda w, x, y, z.w \wedge (x \vee (y \wedge z)), \langle g_{\omega,r}, g_{\chi,r}, g_{\psi,s}, g_{\psi,t} \rangle \rangle$ for $\psi = \chi \,\mathrm{AR}\, \omega$ and r not a leaf,
- $\langle id, g_{\omega,r} \rangle$ for $\psi = \chi \,\mathrm{ER}\, \omega$ or $\psi = \chi \,\mathrm{AR}\, \omega$ and r a leaf,

where $r, s, t \in \tau$, s and t are children of r, and $\psi, \chi, \omega \in \phi$. If r has only one child we set $s = t$ (and merge the dependencies in the resulting circuit into a single dependency) otherwise we require $s \neq t$. Since the construction of $\mathrm{cir}(\phi, \tau)$ is in AC^0 the following lemma provides us with an AC^0-reduction from the problem of deciding $\tau \models \phi$ to the problem of evaluating $\mathrm{cir}(\phi, \tau)$.

Lemma 13. *Given a CTL formula ϕ in positive normal form and a finite computation tree τ. For the circuit $\langle \Gamma, \gamma' \rangle = \mathrm{eval}(\mathrm{cir}(\phi, \tau))$ it holds that*

$$\gamma'(g_{\psi,r}) = \begin{cases} 1 & \text{if } \rho, r \models \psi, \\ 0 & \text{otherwise,} \end{cases}$$

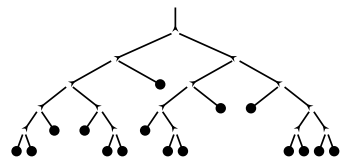
for $\psi \in \mathrm{subf}(\phi)$ and $r \in \tau$.

Proof. We prove the lemma by induction over the structure of ϕ. For $\psi \in \mathrm{subf}(\phi)$ being an atomic proposition the statement follows directly from the definition of γ' and the semantics of CTL. Given that the statement holds for each subformula $\chi \in \mathrm{subf}(\psi)$. If ψ is a disjunction, a conjunction, a $\mathrm{AX}^{\exists}$-, $\mathrm{EX}^{\exists}$-, $\mathrm{AX}^{\forall}$-,or a $\mathrm{EX}^{\forall}$-formula the statement follows from the semantics of the CTL and the definition of γ'. For ψ being an AU-, EU-, AR- or a ER-formula the statement follows from the definition of γ' and the expansion laws of CTL. □

In order to evaluate the circuit $\mathrm{cir}(\phi, \tau)$ we claim that it is a subgraph of $\phi \boxtimes \tau$:

Lemma 14. *Given a formula ϕ in positive normal form and a finite computation tree τ the circuit $\langle \Gamma, \gamma \rangle = \mathrm{cir}(\phi, \tau)$ is a subgraph of $\phi \boxtimes \tau$.*

Proof. Let $\langle V, E \rangle = \phi \boxtimes \tau$. Let $e = \langle \langle \phi, \tau_r \rangle, \langle \psi, \tau_s \rangle \rangle$ an edge in Γ. From the definition of Γ it follows that either $\phi = \psi \wedge s = r + 1$ or $\phi \succ \psi \wedge r = s$ or $\phi \succ \psi \wedge s = r + 1$. Hence e is an edge of $\phi \boxtimes \tau$. □



We are now ready to complete the proof of Theorem 10:

Proof of Theorem 10. Given a CTL formula ϕ and a finite computation tree τ. In L we apply Lemma 12 and then convert ϕ into positive normal form. Use Lemma 8 to AC^0-reduce the problem of deciding $\tau \models \phi$ to the evaluation of $\langle \Gamma, \gamma \rangle = \mathrm{cir}(\phi, \tau)$. By Lemma 14 it holds that $\mathrm{graph}(\Gamma) \subseteq \phi \boxtimes \tau$. Hence we can use Theorem 3 to AC^1-reduce the evaluation of Γ to the evaluation of a circuit that is a subgraph of the normal product of a path and a tree. Using Theorem 4 we get another AC^1 reduction to the evaluation of OIF circuits. Finally, we apply Corollary 1 to evaluate OIF circuits in $\mathsf{logDCFL}$. The overall complexity is $\mathsf{AC}^0(\mathsf{AC}^1(\mathsf{AC}^1(\mathsf{logDCFL}))) = \mathsf{AC}^2(\mathsf{logDCFL})$. □

Chapter 6

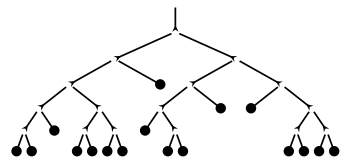
Path Checking for Extensions of LTL

In this chapter we investigate extensions of LTL that are important for practical applications. First, we add past-time modalities to LTL. Second, we extend LTL with bounded modalities that restrict the scope of a subformula to a bounded prefix of the computation path. Both extension are particularly interesting in runtime verification and monitoring, which are the main application domains of finite path checking.

We aim to apply the techniques that we developed for proving that LTL path checking is efficiently parallelizable. As it turns out we can not use these techniques in straightforward way, but we need to extend the approach in different ways. In the end we establish the same upper bound for path checking of the extensions as for pure LTL, namely $\mathsf{AC}^1(\mathsf{logDCFL})$.

6.1 Efficient Path Checking of LTL+Past

Prior's original temporal logic included past-time modalities [80]. On well ordered time domains, as they occur in program verification, the past-time modalities do not add expressive power to the logic [29]. Therefore LTL has been defined as pure future logic. On the other hand the past-time modalities do not add complexity to the model checking problem [85] but allow for more natural expression of properties as formulas [66]. Moreover, there are cases where past-time modalities can even reduce the complexity of model checking, since LTL with past-time modalities can be exponentially more succinct compared to pure future LTL [69]. Therefore Markey concludes that "past is for free"[68]. The benefit of including



past-time modalities into the logic is even more obvious in the context of runtime verification. While the complexity of online monitoring of future formulas is exponential, the complexity drops to only linear for past-time formulas the due to the fact that backward determinization of alternating automata is linear.

Given the practical relevance of LTL with past-time modalities, in this section we extend the path checking techniques that we developed for LTL to LTL with past-time modalities (LTL+Past). We will find that also in path checking *past is for free.*

Theorem 11. *MC[LTL+Past, paths] is in* $\mathsf{AC}^1(\mathsf{logDCFL})$.

Syntax and semantics. We now extend LTL with the *past-time modalities* $\mathrm{Y}^\exists$ (strong yesterday), $\mathrm{Y}^\forall$ (weak yesterday), S (since), and T (trigger) with the following semantics:

- $(\rho, i) \models \mathrm{Y}^\exists \psi$ iff $i - 1 \geq 0 \wedge (\rho, i-1) \models \psi$,
- $(\rho, i) \models \mathrm{Y}^\forall \psi$ iff $i - 1 < 0 \vee (\rho, i-1) \models \psi$,
- $(\rho, i) \models \phi_l \,\mathrm{S}\, \phi_r$ iff $\exists i \geq j \geq 0$ s.t. $(\rho, j) \models \phi_r \wedge \forall i \geq k > j \;.\; (\rho, k) \models \phi_l$, and
- $(\rho, i) \models \phi_l \,\mathrm{T}\, \phi_r$ iff $\forall i \geq j \geq 0 \;.\; (\rho, j) \models \phi_r \vee \exists i \geq k > j$ s.t. $(\rho, k) \models \phi_l$.

The resulting logic is called *linear-time temporal logic with past (LTL+Past)*. We use $\mathrm{P}\,\phi$ (once) to abbreviate $\mathit{true}\,\mathrm{S}\,\phi$ and $\mathrm{H}\,\phi$ (always in the past) to abbreviate $\mathit{false}\,\mathrm{T}\,\phi$.

The following dualities ensure that each LTL+Past formula ϕ can be rewritten into a formula ϕ' in positive normal form with $|\phi'| = O(|\phi|)$.

$$\neg \mathrm{Y}^\forall \phi \;\equiv\; \mathrm{Y}^\exists \neg \phi;$$
$$\neg(\phi_l \,\mathrm{S}\, \phi_r) \;\equiv\; (\neg\phi_l)\,\mathrm{T}(\neg\phi_r).$$

The expansion laws for the past-time modalities are

$$\phi_l \,\mathrm{S}\, \phi_r \;\equiv\; \phi_r \vee (\phi_l \wedge \mathrm{Y}^\exists\,(\phi_l \,\mathrm{S}\, \phi_r));$$
$$\phi_l \,\mathrm{T}\, \phi_r \;\equiv\; \phi_r \wedge (\phi_l \vee \mathrm{Y}^\forall\,(\phi_l \,\mathrm{T}\, \phi_r)).$$

Extended normal product. We would like to prove Theorem 11 in the same way as we proved Theorem 6 for pure future LTL. Consider the formula $\phi = \mathrm{G}\,\mathrm{X}^\exists\,\mathrm{Y}^\exists\,\mathrm{P}\,p$. Figure 6.1 shows the circuit that result from expanding ϕ over some finite computation path ρ. The problem is that the graph of the circuit is not a subgraph of the normal product of the finite computation path ρ and the formula tree ϕ. For the subcircuits that correspond to past-time modalities the

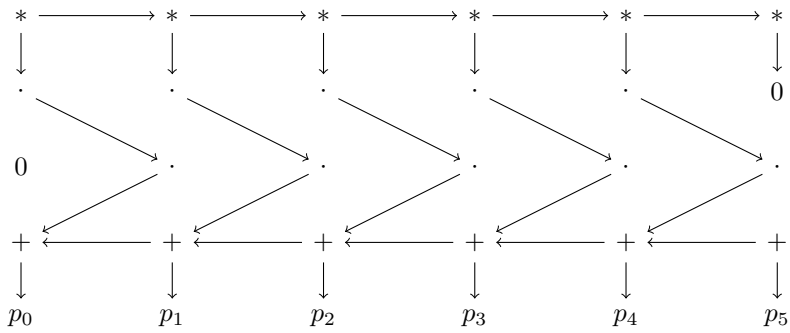


Figure 6.1: The circuit that results from expanding $\mathrm{G}\,\mathrm{X}^{\exists}\,\mathrm{Y}^{\exists}\,\mathrm{P}\,p$ over a computation path ρ with $|\rho| = 6$. Gates labeled with $*$ are and-gates and gates labeled with $+$ are or-gates. Observe that it is not a subgraph of the normal product of the path ρ and the formula tree.

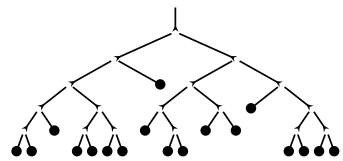
edges have the "wrong" direction. As a consequence for the yesterday-modality the "diagonal" edges use the "wrong" diagonal. Intuitively, the product is built using the $\boxslash$ operator instead of the $\boxbslash$ operator.

We solve this problem by extending the product construction: We define the *extended normal product* on vertex labeled graphs. The labels indicate how to interpret the direction of the edges in the product. We introduce a Boolean flag "*rev*" that, if set for a vertex v, indicates that in the product for each product-vertex $\langle v, w\rangle$ all outgoing edges of w are reversed. For a graph G let $G^{-1} = \langle V(G), E(G)^{-1}\rangle$. Given rev-labeled graphs G, H, the *extended normal product* of G and H, denoted as $G \boxtimes H$, is the graph with $V(G \boxtimes H) = V(G) \times V(H)$ and $e = \langle\langle g, h\rangle, \langle g', h'\rangle\rangle \in E(G \boxtimes H)$ if and only if

- $e \in E(G \boxbslash H)$ for neither $rev(g)$ nor $rev(h)$,
- $e \in E(G^{-1} \boxbslash H)$ for $rev(g)$ and not $rev(h)$,
- $e \in E(G \boxbslash H^{-1})$ for $rev(h)$ and not $rev(h)$, and
- $e \in E(G^{-1} \boxbslash H^{-1})$ for $rev(g)$ and $rev(h)$,

where $g, g' \in V(G)$ and $h, h' \in V(H)$.

Remark: the symbol $\boxtimes$ might suggest that the product of two edges results in a K_4. This would for imply that the product of two paths is in general not planar (a path of length two and another path of length three would suffice have get a K_5 as a subgraph). This is not true. From the definition of $\boxtimes$ it follows



that there is always only one diagonal present. Intuitively, one must not think of $\boxtimes$ as $\boxbslash$ *and* $\boxslash$ but think of $\boxtimes$ as $\boxbslash$ *or* $\boxslash$.

In order to adapt the efficient evaluation of circuits to the new product we state two observations:

1. For two directed acyclic graphs G and H the extended normal product $G \boxtimes H$ is a directed acyclic graph if for either G or H all vertices v are labeled in the same way, and

2. for two paths G and H the extended normal product is planar.

From these observations it is straightforward to conclude that the efficient parallel evaluation algorithm from Theorem 4 can be extended to circuits that are a subgraphs of the extended normal product of a path and a tree where no node of the tree is labeled with *rev*.

Theorem 12. *Given a circuit* Γ *with* $\mathrm{graph}(\Gamma) \subseteq (G \boxtimes H)$ *for some (rooted) tree* G *and some (directed) path* H *where no node of* H *is labeled with rev. The evaluation problem for* Γ *can be* AC^1*-reduced to the evaluation problem for* OIF *circuits with variable gates.* □

Efficient parallel path checking algorithm for LTL+Past. Having established Theorem 12 the proof of Theorem 11 is a straightforward extension of the proof of Theorem 6. As usual, we start with many-one reducing the path checking problem to a circuit evaluation problem.

Given a LTL+Past formula ϕ in positive normal form and a finite computation path $\rho = \rho_0, \ldots, \rho_{n-1}$, we construct the circuit $\mathrm{cir}(\phi, \rho) = \langle \Gamma, \gamma \rangle$ with $\Gamma = \{g_{\psi,\rho_r} \mid \psi \in \phi, 0 \leq r < n\}$ and $\gamma(g_{\psi,\rho_r})$ defined by

- $\langle \mathrm{id}, g_{\chi,\rho_{r-1}} \rangle$ for $\psi = \mathrm{Y}^{\exists}\chi$ and $r > 0$,
- 0 for $\psi = \mathrm{Y}^{\exists}\chi$ and $r = 0$,
- $\langle \mathrm{id}, g_{\chi,\rho_{r-1}} \rangle$ for $\psi = \mathrm{Y}^{\forall}\chi$ and $r > 0$,
- 1 for $\psi = \mathrm{Y}^{\forall}\chi$ and $r = 0$,
- $\langle \lambda x, y, z.x \vee (y \wedge z), \langle g_{\omega,\rho_r}, g_{\chi,\rho_r}, g_{\psi,\rho_{r-1}} \rangle \rangle$ for $\psi = \chi \,\mathrm{S}\, \omega$ and $r > 0$,
- $\langle id, g_{\omega,\rho_0} \rangle$ for $\psi = \chi \,\mathrm{S}\, \omega$ and $r = 0$,
- $\langle \lambda x, y, z.x \wedge (y \vee z), \langle g_{\omega,\rho_r}, g_{\chi,\rho_r}, g_{\psi,\rho_{r-1}} \rangle \rangle$ for $\psi = \chi \,\mathrm{T}\, \omega$ and $r > 0$, and
- $\langle id, g_{\omega,\rho_0} \rangle$ for $\psi = \chi \,\mathrm{T}\, \omega$ and $r = 0$,

where $0 \leq r < n$ and $\psi, \chi, \omega \in \phi$. For the connectives already present in future LTL the construction is described in Section 4.2. The construction of $\mathrm{cir}(\phi, \rho)$ is in AC^0 and the following lemma provides us with an AC^0-reduction from the problem of deciding $\rho \models \phi$ to the problem of evaluating $\mathrm{cir}(\phi, \rho)$. The proof is analogous to the proof of Lemma 8.

Lemma 15. *Given a LTL+Past formula ϕ in positive normal form and a finite computation path ρ. For the circuit $\langle \Gamma, \gamma' \rangle = \mathrm{eval}(\mathrm{cir}(\phi, \rho))$ it holds that*

$$\gamma'(g_{\psi,\rho_r}) = \begin{cases} 1 & \text{if } \rho, r \models \psi, \\ 0 & \text{otherwise,} \end{cases}$$

for $\psi \in \phi$ and $0 \leq r < n$. □

The following lemma about the circuit $\mathrm{cir}(\phi, \rho)$ allows us to apply Theorem 12 in order to evaluate it.

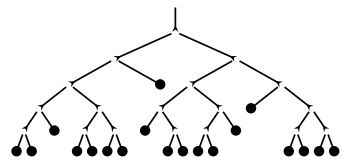
Lemma 16. *Given an LTL+Past formula ϕ in positive normal form and a finite computation path ρ, the circuit $\mathrm{cir}(\phi, \rho)$ is a subgraph of $\phi \boxtimes \rho$ where a node in ϕ is labeled with rev if and only if it is a past-time modality.* □

Proof. Let $e = \langle \langle \phi, \rho_r \rangle, \langle \psi, \rho_s \rangle \rangle$ an edge in $\mathrm{cir}(\phi, \rho)$. From the definition of $\mathrm{cir}(\phi, \rho)$ it follows that either $\phi = \psi \wedge s = r \pm 1$ or $\phi \succ \psi \wedge r = s$ or $\phi \succ \psi \wedge s = r \pm 1$. Hence, e is an edge of $\phi \boxtimes \rho$. □

We are now ready to complete the proof of Theorem 11. It is a straightforward adaption of the proof of Theorem 6 from Section 4.2 about path checking for LTL.

Proof of Theorem 6. Given an LTL+Past formula ϕ and a finite computation path ρ. In L convert ϕ into positive normal form. Use Lemma 15 to AC^0-reduce the problem of deciding $\rho \models \phi$ to the evaluation of $\langle \Gamma, \gamma \rangle = \mathrm{cir}(\phi, \rho)$. By Lemma 16 Γ is a subgraph of the extended normal product $\phi \boxtimes \rho$ where no node of ρ is labeled with *rev*. Hence, we can use Theorem 12 to AC^1-reduce the evaluation of Γ to the evaluation OIF circuits with variable gates. Thus, we can use the algorithm from Corollary 1 which runs in $\mathsf{logDCFL}$. The overall complexity is $\mathsf{AC}^1(\mathsf{logDCFL})$. □

Remark: For CTL tree checking Theorem 3 can not be extended in the same way, because the parallel tree contraction would fail if some child node in the tree depends on its parent. This would violate the condition that each branch of the tree can be evaluated independently of all other branches, thus preventing the possibility to evaluate branches in parallel. This is one reason we cannot tackle tree checking for multi-modal CTL with our approach. Multi-modal CTL would subsume Core XPath which is known to be complete for P. For CTL with past it is an open problem whether tree checking is hard for P or the problem is in NC.



6.2 Efficient Path Checking of BLTL

In practical applications unbounded future modalities are problematic: for the response property $\mathrm{G}(\mathit{request} \rightarrow \mathrm{F}\,\mathit{response})$, when may we expect the *reponse* after a *request* has been issued? Immediately, in some seconds, some minutes, hours, days, or years? Particularly, in online monitoring this poses a problem: if only a finite prefix of a computation is visible, it is impossible to falsify an unbounded liveness property or validate an unbounded safety property. Bounded modalities, that allow quantitative statements over time, offer a solution in such a situation. This observation has led to the introduction of real-time temporal logics [47, 3, 56, 4] that are interpreted over computations paths where each state is stamped with a value from a real-valued time domain, so called *timed state sequences*. In this thesis we interpret the bounds over normal computation paths, i.e. the bounds just count states. This semantics are common in hardware verification [45] where the system under consideration is assumed to be clocked.

In this section we will extend the path checking techniques for LTL+Past to LTL with bounded modalities (BLTL+Past). We will find that in path checking *bounds are for free.*

Theorem 13. *MC[BLTL+Past, paths] is in* $\mathsf{AC}^1(\mathsf{logDCFL})$

Syntax and semantics. To obtain *linear-time temporal logic with past and bounds (BLTL)* we further add the bounded temporal operators U_b, R_b, S_b, and T_b, where $b \in \mathbb{N}$ is any natural number. For technical reasons we define for the remainder of the section the *size of a formula* using unary encoding for the bounds. Note, however, that Theorem 13 also holds for the usual $O(1)$-encoding of the bounds. The semantics of the bounded operators is defined as follows:

- $(\rho, i) \models \phi_l \,\mathrm{U}_b\, \phi_r$ iff $\exists i \leq j \leq \min(i+b, |\rho|-1)$ s.t $(\rho, j) \models \phi_r \wedge \forall i \leq k < j$, $(\rho, k) \models \phi_l$,
- $(\rho, i) \models \phi_l \,\mathrm{R}_b\, \phi_r$ iff $\forall i \leq j \leq \min(i+b, |\rho|-1), (\rho, j) \models \phi_r \vee \exists i \leq k < j$ s.t $(\rho, k) \models \phi_l$,
- $(\rho, i) \models \phi_l \,\mathrm{S}_b\, \phi_r$ iff $\exists i \geq j \geq \max(i-b, 0)$ s.t $(\rho, j) \models \phi_r \wedge \forall i \geq k > j$, $(\rho, k) \models \phi_l$, and
- $(\rho, i) \models \phi_l \,\mathrm{T}_b\, \phi_r$ iff $\forall i \geq j \geq \max(i-b, 0), (\rho, j) \models \phi_r \vee \exists i \geq k > j$ s.t $(\rho, k) \models \phi_l$.

The following dualities apply for the BLTL operators:

$$\neg(\phi_l \,\mathrm{U}_b\, \phi_r) \equiv (\neg\phi_l)\,\mathrm{R}_b\,(\neg\phi_r)\ ;$$
$$\neg(\phi_l \,\mathrm{S}_b\, \phi_r) \equiv (\neg\phi_l)\,\mathrm{T}_b\,(\neg\phi_r)\ .$$

The expansion laws for the bounded operators are for $b \in \mathbb{N}$

$$\phi_l \,\mathrm{U}_b\, \phi_r \equiv \begin{cases} \phi_r \vee (\phi_l \wedge \mathrm{X}^{\exists}(\phi_l \,\mathrm{U}_{b-1}\, \phi_r)) & \text{for } b > 0, \\ \phi_r & \text{for } b = 0, \end{cases}$$

$$\phi_l \,\mathrm{R}_b\, \phi_r \equiv \begin{cases} \phi_r \wedge (\phi_l \vee \mathrm{X}^{\forall}(\phi_l \,\mathrm{R}_{b-1}\, \phi_r)) & \text{for } b > 0, \\ \phi_r & \text{for } b = 0, \end{cases}$$

$$\phi_l \,\mathrm{S}_b\, \phi_r \equiv \begin{cases} \phi_r \vee (\phi_l \wedge \mathrm{Y}^{\exists}(\phi_l \,\mathrm{S}_{b-1}\, \phi_r)) & \text{for } b > 0, \\ \phi_r & \text{for } b = 0, \text{ and} \end{cases}$$

$$\phi_l \,\mathrm{T}_b\, \phi_r \equiv \begin{cases} \phi_r \wedge (\phi_l \vee \mathrm{Y}^{\forall}(\phi_l \,\mathrm{T}_{b-1}\, \phi_r)) & \text{for } b > 0, \\ \phi_r, & \text{for } b = 0. \end{cases}$$

In the remainder of this section we will present a path checking algorithm for BLTL. The algorithm constructs a circuit that is of polynomial size in the size of the input formula, the length of the input computation path, and the sum of the bounds that occur in the input formula. However, we do not want the complexity of the algorithm to depend on the encoding of the bounds. The following theorem allows us to prune the size of the bounds that occur in a BLTL formula to the length of the computation path.

Theorem 14. *Given a BLTL formula ϕ and a finite computation path ρ. The BLTL formula ϕ' is obtained from ϕ by setting each bound n in ϕ to* $\min(n, |\rho|)$. *It holds that $\rho \models \phi$ if and only if $\rho \models \phi'$.*

Proof. By induction over ϕ. □

In the following we always assume that any bound occurring in a BLTL formula of a path checking problem has most the size of the computation path. The sum of the bounds is thus polynomial in the size of the input formula and the input computation path. Therefore we do not need to consider the bounds in the complexity analysis.

Boolean circuits for BLTL path checking. As usual we prove this theorem by first constructing a circuit from the input formula and the computation path. We then show that we can evaluate the circuit in $\mathsf{AC}^1(\mathsf{logDCFL})$. However, the expansion laws for the bounded modalities yield a circuit that is not a subgraph of the extended normal product of the formula tree and path. During the recursive expansion gives rise to "new" subformulas that where not present in the original formula: An U_n-operator gives rise to "additional" U_i-operators for $0 < i < n$. Let ϕ a BLTL formula and ρ a computation path. Let Φ the set of all subformulas that occur in the expansion while construction the circuit $\mathrm{cir}(\phi, \rho)$. Then the

decomposition $\mathrm{cir}(\phi,\rho)/\equiv_\Phi$ is not a tree but a generally a directed acyclic graph as shown in Figure 6.2. This prevents us from using Theorem 12 for evaluating $\mathrm{cir}(\phi,\rho)$. The reason for $\mathrm{cir}(\phi,\rho)/\equiv_\Phi$ not being planar are the "new" subformulas in Φ that are not present in ϕ.

Let $\chi \in \mathrm{subf}(\phi)$ be bounded by n, i.e. the top-level modality is a bounded modality with bounded n. Let Ψ be the set of all subformulas in $\Phi \setminus \preceq(\phi)$ that originated from the expansion of χ. For each $\psi \in \Psi$ and each position ρ_i of ρ we will merge the the gates $\langle \psi, \rho_i \rangle$ into the gate $\langle \chi, \rho_i \rangle$. Since $\Phi(\setminus(\Phi \setminus \mathrm{subf}(\phi))) = \mathrm{subf}(\phi)$ we have eliminated all gates from the circuit that correspond to "new" subformulas. As result we get that $\mathrm{cir}(\phi,\rho)/\equiv_{\mathrm{subf}(\phi)}$ is a tree.

What is the labeling of the merged gates $\langle \chi, \rho_i \rangle$? We can use standard β-reduction from the λ-calculus to fold the functions of all involved gates into a single function. However, in the original circuit a gate the ψ gates depend on each other on different path positions. E.g. a gate $\langle \psi, \rho_i \rangle$ might depend on a gate $\langle \psi, \rho_{i+1} \rangle$. Therefore we need some means to access the intermediate values that would be hidden by the β-reduction. The solution is to extend the type of the gates to represent the values of all subsumed gates: for a path position ρ_i the merged gate $\langle \chi, \rho_i \rangle$ does compute the value of the original gate $\langle \chi, \rho_i \rangle$ along with the values of all gates $\langle \psi, \rho_i \rangle$ for all $\psi \in \Psi$. Hence, the merged gate $\langle \chi, \rho_i \rangle$ does not compute a Boolean value but vector of Boolean values. Thus, at each position of ρ we merge all ψ-gates into the corresponding χ-gate pass the vector of the values of the original gates along a single dependency to the merged gate at the previous position of ρ. Figure 6.3 illustrates the result of merging the gates the resulted from expanding a bounded modality. By this construction we get circuit that is a subgraph of the extended normal product of the formula tree and the computation path.

In the following we call a gate that represents a Boolean vector an *extended gate*. For convenience we define that a non-extended gate that depends on an extended gate reads just the first bit of the Boolean vector of the extended gate. For a vector $\vec{z} \in \mathbb{B}^n$ the right 0-shift $(0 \gg \vec{z})$ is defined as $\vec{z} \cdot (\delta_{i+1,j})_{0 \le i,j < n}$, where δ is the Kroneker-δ. Analogously the right 1-shift $(1 \gg \vec{z})$ is defined as $1 - \Big((1 - \vec{z}) \cdot (\delta_{i+1,j})_{0 \le i,j < n}\Big)$. We will use the following functions on Boolean vectors in the definition of our circuits: constant $\vec{1}$, constant $\vec{0}$, scalar multiplication with $\vec{1}$ (n-identity), right 0-shift $(0 \gg)$, and right 1-shift $(1 \gg)$.

Given a BLTL formula ϕ in positive normal form and a finite computation path $\rho = \rho_0, \ldots, \rho_{n-1}$, we construct the circuit $\mathrm{cir}(\phi,\rho) = \langle \Gamma, \gamma \rangle$ with $\Gamma = \{g_{\chi,\rho_r} \mid \chi \in \phi, 0 \le r < n\}$ and $\gamma(g_{\chi,\rho_r})$. Definition of $\gamma(g_{\chi,\rho})$ is exactly the same as in the constructions for LTL and LTL+Past except for the cases where the top-level operator of χ is a bounded operator. For $\chi = \psi \,\mathrm{U}_n\, \omega$, $n \in \mathbb{N}$ we

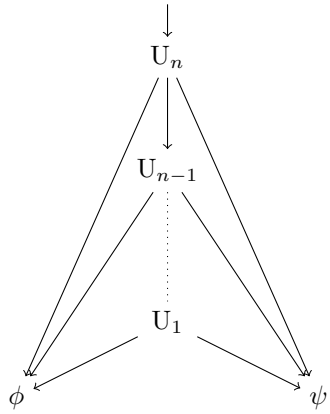


Figure 6.2: Expanding a formula $\chi \,\mathrm{U}_n\, \psi$ over a finite computation path and then projecting onto the formula component reveals that the resulting combinatorial circuit is not a normal product of a path and a tree, but it is the product of a path and a directed acyclic path.

define

$$\gamma(g_{\chi,\rho_r}) = \left\langle f, \left\langle g_{\omega,\rho_r}, g_{\psi,\rho_r}, g_{\chi,\rho_{r+1}} \right\rangle \right\rangle,$$

where $f\colon \mathbb{B} \times \mathbb{B} \times \mathbb{B}^n \to \mathbb{B}^n$ with

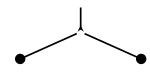
$$f(x, y, \vec{z}) = \begin{cases} \vec{1} & \text{if } x = 1, \\ \vec{0} & \text{if } x = 0 \text{ and } y = 0, \text{ and} \\ 0 \gg \vec{z} & \text{otherwise,} \end{cases} \tag{6.1a}$$

for $0 \leq r < |\rho| - 1$ and

$$\gamma(g_{\chi,\rho_r}) = \left\langle \lambda x.x \cdot \vec{1}, g_{\omega,\rho_r} \right\rangle, \tag{6.1b}$$

for $r = |\rho| - 1$.

Intuitively, the output vector of the gate g_{χ,ρ_r} counts in a backward direction how many steps (into the past) the formula ω has held given that ψ held throughout all these steps. The gate resets the vector to $\vec{1}$ if ω evaluates to 1 at the current position (r). It resets the vector to $\vec{0}$ if ω and ψ both evaluated to 0 at the current position. Otherwise, if ψ holds and ω does not hold it decreases the count by right-shifting the vector. Figure 6.3 illustrates how the U_n-operator is translated into a circuit.



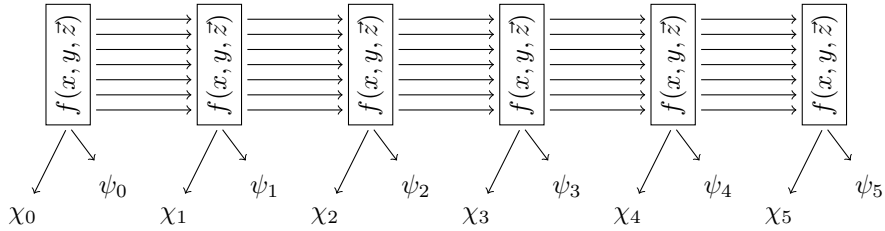


Figure 6.3: Circuit that results from expanding the formula $\chi \,\mathrm{U}_7\, \psi$ over a computation path ρ with $|\rho| = 6$.

The construction for R_n, S_n, and T_n are analogous: For $\psi = \chi \,\mathrm{R}_n\, \omega$ the function f is dual to the U_n case:

$$f(x, y, \vec{z}) = \begin{cases} \vec{0} & \text{if } x = 0, \\ \vec{1} & \text{if } x = 1 \text{ and } y = 1, \text{ and} \\ 1 \gg \vec{z} & \text{otherwise,} \end{cases}$$

for $0 \leq r < |\rho| - 1$ and

$$\gamma(g_{\psi,\rho_r}) = \left\langle \lambda x.x \cdot \vec{1}, g_{\omega,\rho_r} \right\rangle,$$

for $r = |\rho| - 1$. The past operators are defined analogously to the future operators with the order of the dependencies in the ρ-component reversed: For $\chi = \psi \,\mathrm{S}_n\, \omega$ let

$$\gamma(g_{\chi,\rho_r}) = \left\langle f, \left\langle g_{\omega,\rho_r}, g_{\psi,\rho_r}, g_{\chi,\rho_{r-1}} \right\rangle \right\rangle,$$

where $f \colon \mathbb{B} \times \mathbb{B} \times \mathbb{B}^n \to \mathbb{B}^n$ with

$$f(x, y, \vec{z}) = \begin{cases} \vec{1} & \text{if } x = 1, \\ \vec{0} & \text{if } x = 0 \text{ and } y = 0, \text{ and} \\ 0 \gg \vec{z} & \text{otherwise,} \end{cases}$$

for $0 < r < |\rho|$ and

$$\gamma(g_{\chi,\rho_r}) = \left\langle \lambda x.x \cdot \vec{1}, g_{\omega,\rho_r} \right\rangle,$$

for $r = 0$. Finally, for $\chi = \psi \,\mathrm{T}_n\, \omega$ the function f is dual to the S_n case:

$$f(x, y, \vec{z}) = \begin{cases} \vec{0} & \text{if } x = 0, \\ \vec{1} & \text{if } x = 1 \text{ and } y = 1, \text{ and} \\ 1 \gg \vec{z} & \text{otherwise,} \end{cases}$$

for $0 < r < |\rho|$ and

$$\gamma(g_{\chi,\rho_r}) = \left\langle \lambda x.x \cdot \vec{1}, g_{\omega,\rho_r} \right\rangle,$$

for $r = 0$. The circuit $\mathrm{cir}(\phi, \rho)$ can be constructed in L.

Lemma 17. *Given a BLTL formula ϕ in positive normal form and a finite computation path ρ. For the circuit $\langle \Gamma, \gamma' \rangle = \mathrm{eval}(\mathrm{cir}(\phi, \rho))$ it holds that*

$$\gamma'(g_{\chi,\rho_r}) = \begin{cases} 1 & \text{if } \rho, r \models \chi, \\ 0 & \text{otherwise,} \end{cases}$$

for $\chi \in \phi$ and $0 \leq r < n$.

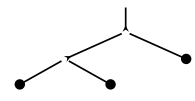
Proof. Proof by induction over ϕ. For all but the bounded connectives the correctness follows from the proof of Lemma 8. For the bounded operators it is a consequence of the definition of γ (in particular f) and the semantics of BLTL. □

In order to evaluate the circuit $\mathrm{cir}(\phi, \rho)$ we show that it is a subgraph of the extended normal product $\phi \boxtimes \rho$. The following lemma can be proved straightforward by checking all dependencies between gates in the construction of the circuit.

Lemma 18. *Given a BLTL formula ϕ in positive normal form and a finite computation path ρ the circuit $\langle \Gamma, \gamma \rangle = \mathrm{cir}(\phi, \rho)$ is a subgraph of $\phi \boxtimes \rho$.* □

An efficient parallel path checking algorithm for BLTL. We would like to use Theorem 12 to evaluate Γ. However, that theorem talks about circuits that do not contain extended gates. There are two principle problems in extending the evaluation techniques from the previous chapters to extended circuits: First, we can not expect to perform the evaluation with constant resources if the vectors are of non-constant size and second, it is not clear how to lift the notion of monotonicity to Boolean vectors such that we can profit from the low complexity of the evaluation of monotone planar circuits. In the following we will solve this problems and provide a version of the path checking algorithm for LTL/ LTL+Past that works for BLTL as well.

Except for the evaluation algorithm that used for the base case, namely for the evaluation of monotone planar circuits, the proof of Theorem 12 is agnostic of the distinction between normal and extended gates . In order to apply Theorem 12 we translate all extended gates back into normal gates just before we call the oracle for the evaluation of monotone planar circuits. Unfortunately, there is no local translation from extended gates into normal gates such that the resulting circuit is still the extended normal product of two paths. Our translation from



extended into normal gates will therefore use global knowledge about the circuit and the evaluation algorithm. Namely, we will use that fact that all extended gates corresponding to the same bounded connective in the formula tree are translated back into normal gates simultaneously. We now proceed with the proof of Theorem 13.

Proof of Theorem 13. Given an BLTL formula ϕ and a finite computation path ρ, in L convert ϕ into positive normal form, use Lemma 8 to L-reduce the problem of deciding $\rho \models \phi$ to the evaluation of $\langle \Gamma, \gamma \rangle = \mathrm{cir}(\phi, \rho)$. By Lemma 9 Γ is a subgraph $\phi \boxtimes \rho$.

We now show how we can adapt Theorem 12 in order to evaluate Γ in $\mathsf{AC}^1(\mathsf{logDCFL})$. To simplify the following explanations assume that there are no past-time modalities in ϕ. As mentioned before the proof of Theorem 4 does not depend on the distinction between normal and extended gates except for the step where the oracle for the evaluation of OIF circuits is called. Therefore it suffices to show that always when the algorithm calls the oracle for the evaluation of an OIF subcircuit we can translate this extended subcircuit into an equivalent normal circuit.

The algorithm in the proof of Theorem 4 calls the oracle only on a circuit $\langle \mathsf{B}, \beta \rangle$ that is a subgraph of the extended normal product of ρ and some path P in the formula tree. Consider some extended gate g in B. Let $\varphi \in \mathrm{subf}(\phi)$ be the bounded subformula with $\varphi \in P$ such that g is of the form g_{φ,ρ_r} for some $0 \leq r < |\rho|$. For each $0 \leq r < |\rho|$ it holds that g_{φ,ρ_r} is an extended gate and $g_{\varphi,\rho_r} \in \mathsf{B}$. $\mathsf{B} \subseteq \rho \boxtimes P$ implies that there is a *fixed* subformula $\psi \prec \varphi$ such that for all $0 \leq r < |\rho|$ the only possible dependencies are g_{ψ,ρ_r} or $g_{\varphi,\rho_{r+1}}$. In other words, if in the original circuit Γ we have

$$\gamma(g_{\varphi,\rho_r}) = \begin{cases} \langle \lambda xy\vec{z}.f(x,y,\vec{z}), \langle g_{\psi,\rho_r}, g_{\chi,\rho_r}, g_{\varphi,\rho_{r+1}} \rangle \rangle & \text{for } 0 < r < |\rho - 1| \text{ and} \\ \langle \lambda x.g(x), g_{\psi,\rho_r} \rangle & \text{for } r = |\rho| - 1 \end{cases}$$

then in the subcircuit B that is presented to the oracle we have either

$$\beta(g_{\varphi,\rho_r}) = \begin{cases} \langle \lambda y\vec{z}.f(C,y,\vec{z}), \langle g_{\chi,\rho_r}, g_{\varphi,\rho_{r+1}} \rangle \rangle & \text{for } 0 < r < |\rho - 1| \text{ and} \\ C & \text{for } r = |\rho| - 1 \end{cases} \tag{6.2}$$

or

$$\beta(g_{\varphi,\rho_r}) = \begin{cases} \langle \lambda x\vec{z}.f(x,C,\vec{z}), \langle g_{\psi,\rho_r}, g_{\varphi,\rho_{r+1}} \rangle \rangle & \text{for } 0 < r < |\rho - 1| \text{ and} \\ \langle \lambda x.g(x), g_{\psi,\rho_r} \rangle & \text{for } r = |\rho| - 1 \end{cases} \tag{6.3}$$

where $C \in \mathbb{B}$ depends on r. Moreover, the proof of Theorem 4 decomposes the circuit Γ only in such a way that for φ always all the gates $g_{\varphi,\rho_r}, 0 \leq r < |\rho|$ (in

the following called φ-gates) are in the same subcircuit. Therefore the φ-gates are always evaluated all together and in particular are evaluated for the first time in the same oracle call. Hence, the φ-gates are translated back into normal gates simultaneously. This justifies that the above two cases for B are complete and that we can in the following provide a single global circuit construction for all φ-gates.

The proof will proceed as follows: In a first step we will, for a bounded formula φ and each of the cases (6.2) and (6.3), translate the extended gates individually into a series of normal gates. In a second step we will argue that the circuit obtained by substituting the extended gates with its implementations can be transformed into an equivalent circuit that is a subgraph of the extended normal product of ρ and a path obtained from the formula path by splitting the node of the subformula φ into subpaths of nodes corresponding to the newly introduced normal gates.

Consider the BLTL formula $\varphi = \chi \,\mathrm{U}_n\, \psi$. Recall that the function f in the labeling of g_{φ,ρ_r} is defined by

$$f(x, y, \vec{z}) = \begin{cases} \vec{1} & \text{if } x = 1, \\ \vec{0} & \text{if } x = 0 \text{ and } y = 0, \text{ and} \\ 0 \gg \vec{z} & \text{otherwise,} \end{cases}$$

for $0 \leq r < |\rho| - 1$, where the variable x corresponds to the ψ-gates and y corresponds to the χ-gates. For $r = |\rho| - 1$ g_{φ,ρ_r} only depends on ψ. We define g by $g(x) = x \cdot \vec{1}$.

We begin with the construction for the case (6.2): From (6.1) and (6.2) we deduce for f and g the functions $f_{x=C}$ and $g_{x=C}$ by binding parameter x with $C \in \mathbb{B}$:

$$f_{x=0}(y, \vec{z}) = \begin{cases} \vec{0} & \text{if } y = 0 \text{ and} \\ 0 \gg \vec{z} & \text{if } y = 1, \end{cases}$$

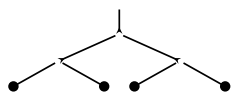
for $x = 0$ and

$$f_{x=1}(y, \vec{z}) = \vec{1}$$

for $x = 1$, and $g_{x=0} = \vec{0}$ and $g_{x=1} = \vec{1}$.

We implement the functions $f_{x=0}$, $f_{x=1}$, $g_{x=0}$, and $g_{x=1}$ as follows. In place of g_{φ,ρ_r} we introduce the gates g_{φ_i,ρ_r} for $0 \leq i < n$ into B. The gates for the functions $f_{x=0}$ and $g_{x=0}$ are labeled

$$\beta(g_{\varphi_i,\rho_r}) = \begin{cases} 0 & \text{for } r = |\rho| - 1 \text{ or } i = 0 \text{ and} \\ \langle \wedge, \langle g_{\chi,\rho_r}, g_{\varphi_{i-1},\rho_{r+1}} \rangle \rangle & \text{for } r < |\rho| - 1 \text{ and } 0 < i < n. \end{cases}$$



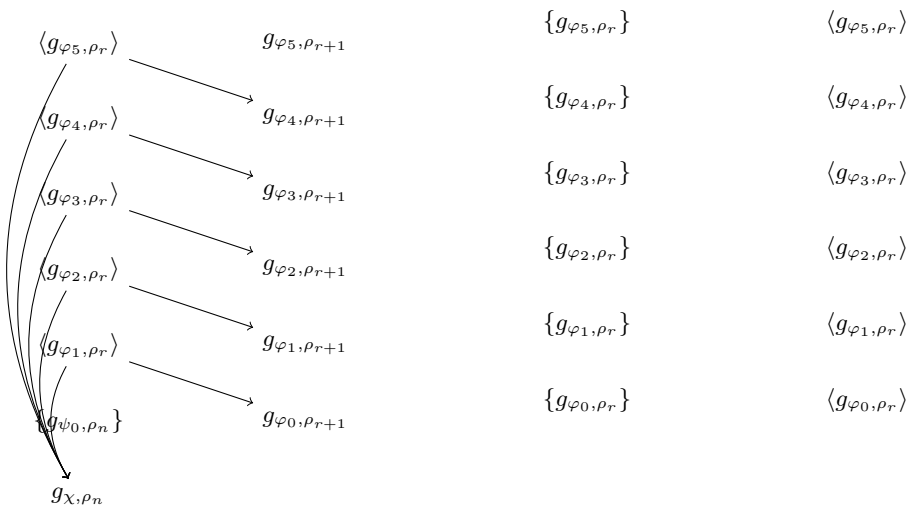


Figure 6.4: Circuit with normal gates resulting from a U_6-gate. The case of $f_{x=0}, r < |\rho| - 1$ is shown on the left, the case for $f_{x=0}, r = |\rho| - 1$ in the middle, and the case of $f_{x=1}$ on the right side. Gates in angle brackets are and-gates. And-gates without dependency are 1. Gates in curly brackets are or-gates. Or-gates without dependency are 0.

For the functions $f_{x=1}$ and $g_{x=1}$ the gates are labeled

$$\beta(g_{\varphi_i,\rho_r}) = 1 \qquad \text{for } 0 \leq i < n \text{ and } 0 \leq r < |\rho|.$$

In order to embed the new gates into B we set $g_{\varphi,\rho_r} = g_{\varphi_{n-1},\rho_r}$ for $0 \leq r < |\rho|$. Figure Figure 6.4 shows for $n = 6$ the respective circuits for $f_{x=0}$ and $f_{x=1}$. The intuition behind the construction is that gate $g_{\varphi_i,\rho_r}, 0 \leq i < n$ is equivalent to the value of the i^{th} component of the value of the original gate g_{φ,ρ_r}. Thus the circuit B with the new gates is equivalent to the original circuit. In Figure 6.5 the circuit B is shown for $n = 6$, $|\rho| = 8$ with $f_{x=1}$ at position 4 and 7 and $f_{x=0}$ at all other positions i. Observe that the circuit is not a subgraph of an extended normal product of two paths. It is not even planar as illustrated by the red colored overlay that is a $K_{3,3}$ in Figure 6.6.

Can we transform the circuit B such that it becomes an extended normal product of ρ and a path? Many gates in B are equivalent to 0 as shown in Figure 6.7: since all gates are either and-gates or constants all gates that depend on a 0-gate are equivalent to 0. We can use this observation to transform the circuit into an equivalent circuit where we set all the gates that depend on a 0-gate to

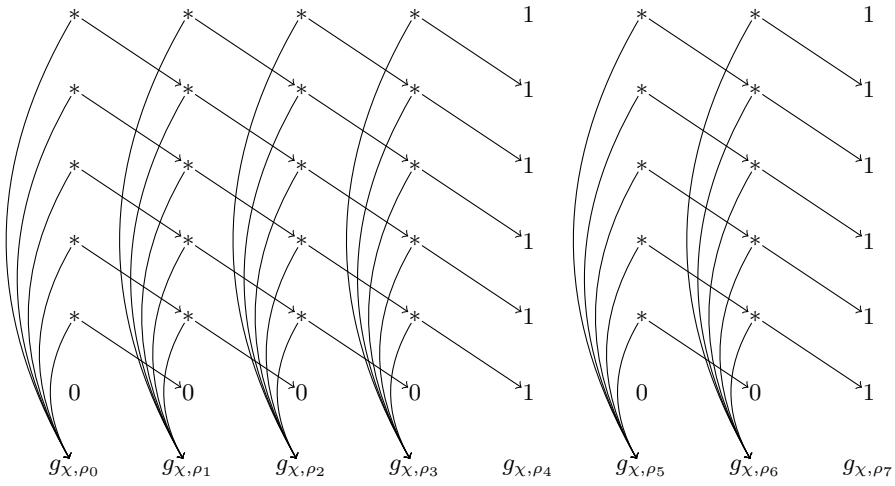


Figure 6.5: The circuit construction for the formula $\chi \,\mathrm{U}_6\, \psi$ over a computation path ρ with $|\rho| = 8$ and $f_{x=1}$ at position 4 and 7 and $f_{x=0}$ at all other positions i. A $*$ denotes an and-gate.

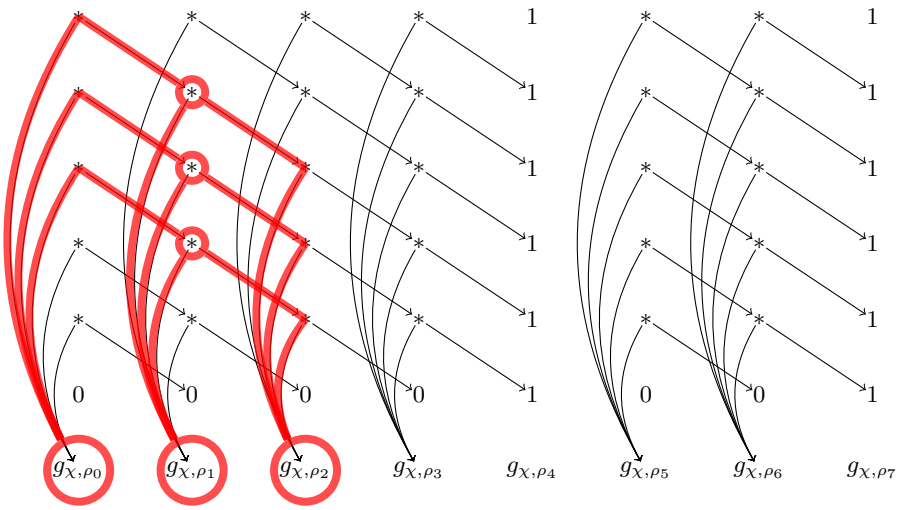
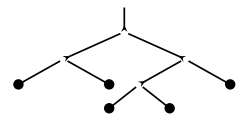


Figure 6.6: The same circuit as in Figure 6.5. The red subgraph is a $K_{3,3}$ that illustrates that the graph is not planar.



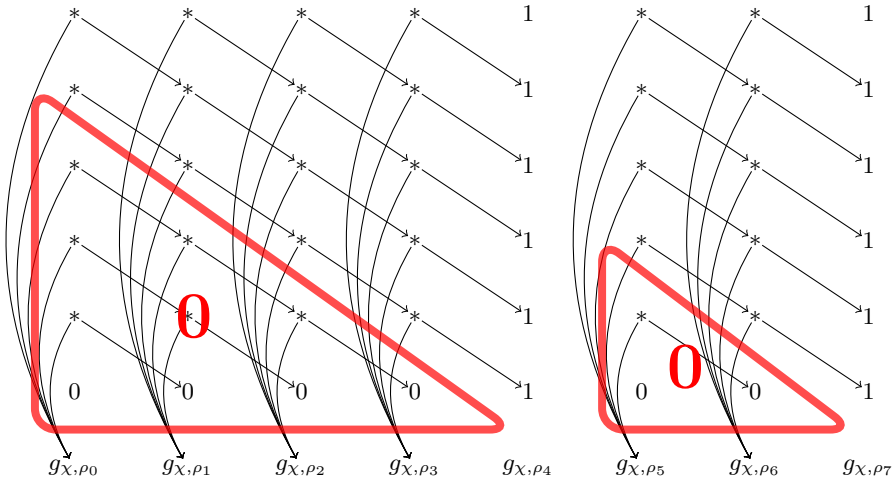


Figure 6.7: The same circuit as in Figure 6.5. All gates are either and-gates or constant gates. Hence all gates that depend on a 0-gate are equivalent to 0.

0. Additionally we can change the dependencies of the remaining and-gates as shown in Figure 6.8. In a last step we reinsert the "old" 0-gates as identity-gates. In a strict sense the resulting circuit is not equivalent to the original circuit B, because we changed the meaning of the 0-gates. However, the top level-gates remain equivalent and all external depencies are to the top-level gates. This last step achieves that the result is a subgraph of the extended normal product of ρ and a path. Figure 6.9 shows the final circuit B.

Summing up, the final circuit B is defined as follows. For g_{φ,ρ_r} in Γ labeled with $f_{x=0}$ or $g_{x=0}$ let r^0 the minimal r' such that $g_{\varphi,\rho_{r'}}$ is labeled with $f_{x=1}$ or $r^0 = \infty$ if there is no such r'. We have

$$\beta(g_{\varphi_i,\rho_r}) = \begin{cases} 0 & \text{for } r = |\rho| - 1, \\ \langle \mathrm{id}, g_{\varphi_{i-1},\rho_r} \rangle & \text{for } r < |\rho| - 1 \text{ and } 0 < i < r' - r, \\ \langle \wedge, \langle g_{\varphi_{i-1},\rho_r}, g_{\varphi_{i-1},\rho_{r+1}} \rangle \rangle & \text{for } r < |\rho| - 1 \text{ and } r' - r \leq i < n, \text{ and} \\ \langle \mathrm{id}, g_{\chi,\rho_r} \rangle & \text{for } r < |\rho| - 1 \text{ and } i = 0 \end{cases}$$

where $\infty - r = \infty$ for all $r \in \mathbb{N}$. It holds that $r' - r > 0$. For g_{φ,ρ_r} in Γ labeled with $f_{x=1}$ or $g_{x=1}$ we have

$$\beta(g_{\varphi_i,\rho_r}) = 1 \qquad \text{for } 0 \leq i < n \text{ and } 0 \leq r < |\rho|.$$

The new gates are embedded into B by setting $g_{\varphi,\rho_r} = g_{\varphi_{n-1},\rho_r}$ for $0 \leq r < |\rho|$. It can easily be checked that B is the extended normal product of the computation

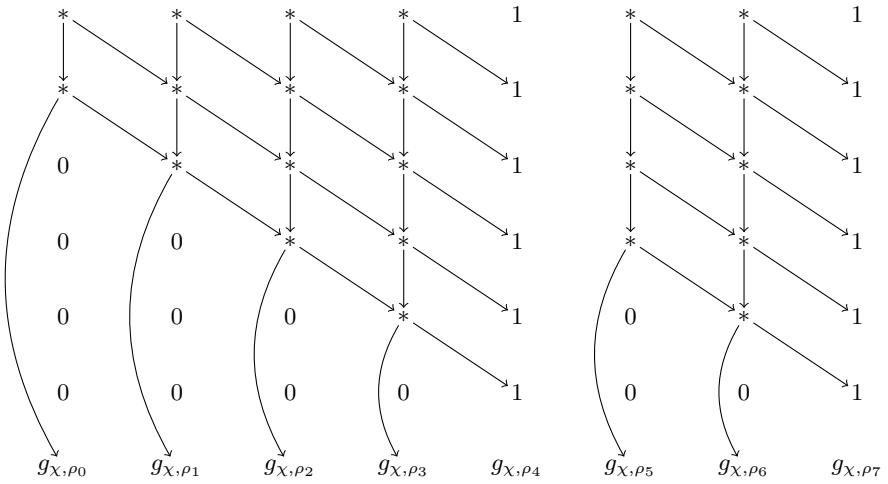


Figure 6.8: A circuit that is obtained from circuit as in Figure 6.7 by propagating the 0-constants and changing the dependencies of the remaining and-gates. Observe that the circuit is equivalent to the circuit from Figure 6.7.

path ρ and the path in the formula where the φ node is replaced by a path $\varphi_0, \ldots, \varphi_{n-1}$.

We have constructed the circuit for the case (6.2). We now consider case (6.3), i.e. we assume that the y variables are bound to either 0 or 1 in the definition of f. From (6.1) and (6.3) we deduce for f and g the functions $f_{y=C}$ and $g_{y=C}$:

$$f_{y=0}(x, \vec{z}) = x \cdot \vec{1}$$

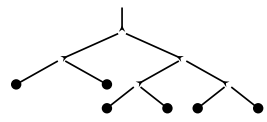
for $y = 0$ and

$$f_{y=1}(x, \vec{z}) = \begin{cases} 0 \gg \vec{z} & \text{if } x = 0 \text{ and} \\ \vec{1} & \text{if } x = 1 \end{cases}$$

for $y = 1$. Further we have $g_{y=0}(x) = g_{y=1}(x) = x \cdot \vec{1}$.

We implement the functions $f_{y=0}$, $f_{y=1}$, $g_{y=0}$, and $g_{y=1}$ as follows. In place of g_{φ,ρ_r} we introduce the gates g_{φ_i,ρ_r} for $0 \leq i < n$ into B. The gates for the functions $f_{y=1}$ and $g_{y=1}$ are labeled

$$\beta(g_{\varphi_i,\rho_r}) = \begin{cases} \langle \vee, g_{\psi,\rho_r} \rangle & \text{for } r = |\rho| - 1 \text{ and} \\ \langle \vee, \langle g_{\psi,\rho_r}, g_{\varphi_{i-1},\rho_{r+1}} \rangle \rangle & \text{for } r < |\rho| - 1, \end{cases}$$



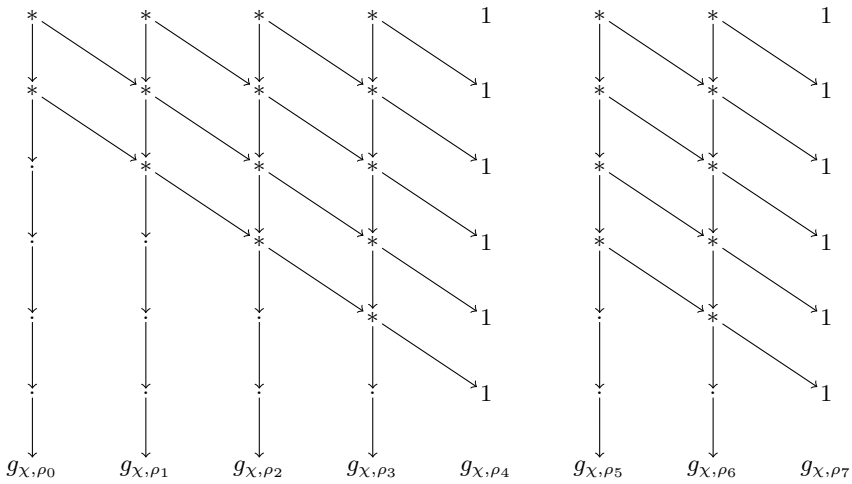


Figure 6.9: The final circuit B. In a strict sense it is not equivalent to Figure 6.8 because the meaning of the former 0-gates changes. However, here it is only important that the top level gates remain equivalent. The circuit clearly is a subgraph of the extended normal product of ρ and a path. Gates denoted with a dot are identity-gates.

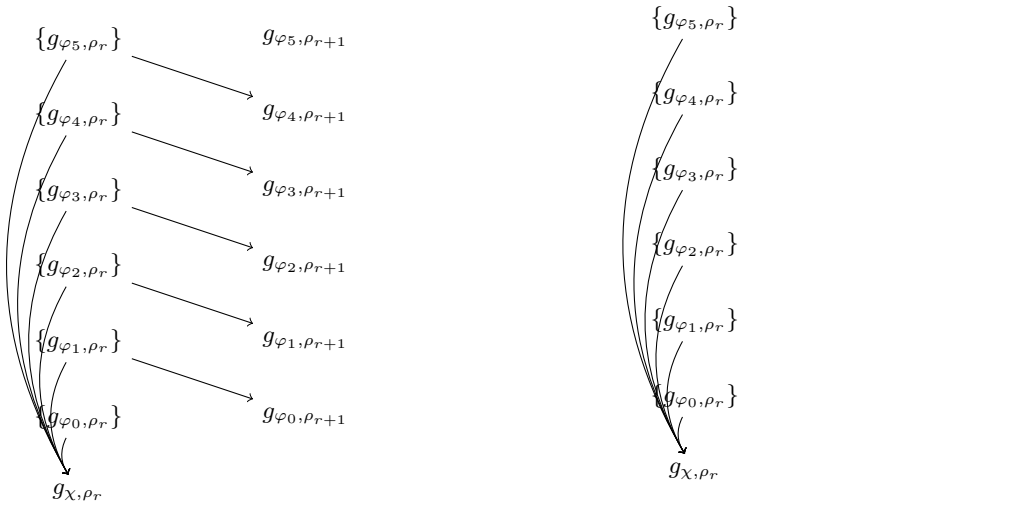


Figure 6.10: Circuits with normal gates resulting from a U_6-gate. The case of $f_{y=1}, r < |\rho| - 1$ is shown on the left. The right side shows all remaining cases. Gates in curly brackets are or-gates. Or-gates without dependency are 0.

for $0 \leq i < n$. For the functions $f_{y=0}$ and $g_{y=0}$ the gates are labeled

$$\beta(g_{\varphi_i,\rho_r}) = \langle \vee, g_{\psi,\rho_r} \rangle \qquad \text{for } 0 \leq i < n \text{ and } 0 \leq r < |\rho|.$$

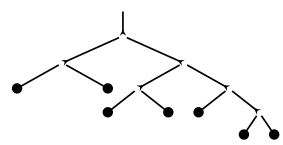
In order to embed the new gates into B we set $g_{\varphi,\rho_r} = g_{\varphi_{n-1},\rho_r}$ for $0 \leq r < |\rho|$. Figure Figure 6.10 shows the respective circuits for f^0_ψ and f^1_ψ for $n = 5$.

In the current case a local translation of the extended gates is already possible. From the definition of the labeling as well as from Figure 6.10 one can easily confirm that the circuits shown in Figure 6.11 are equivalent to the circuits resulting from the previous definition of β. Formally we redefine the labeling β as follows: The gates for the functions $f_{y=1}$ and $g_{y=1}$ are labeled

$$\beta(g_{\varphi_i,\rho_r}) = \begin{cases} \langle \vee, g_{\varphi_{i-1},\rho_r} \rangle & \text{for } r = |\rho| - 1 \text{ and } 1 < i < n, \\ \langle \vee, \langle g_{\varphi_{i-1},\rho_r}, g_{\varphi_{i-1},\rho_{r+1}} \rangle \rangle & \text{for } r < |\rho| - 1 \text{ and } 1 < i < n, \text{ and} \\ \langle \vee, g_{\psi,\rho_r} \rangle & \text{for } i = 0. \end{cases}$$

For the functions $f_{y=0}$ and $g_{y=0}$ the gates are labeled

$$\beta(g_{\varphi_i,\rho_r}) = \begin{cases} \langle \vee, g_{\varphi_{i-1},\rho_r} \rangle & 1 < i < n \text{ and} \\ \langle \vee, g_{\psi,\rho_r} \rangle & \text{for } i = 0 \end{cases}$$



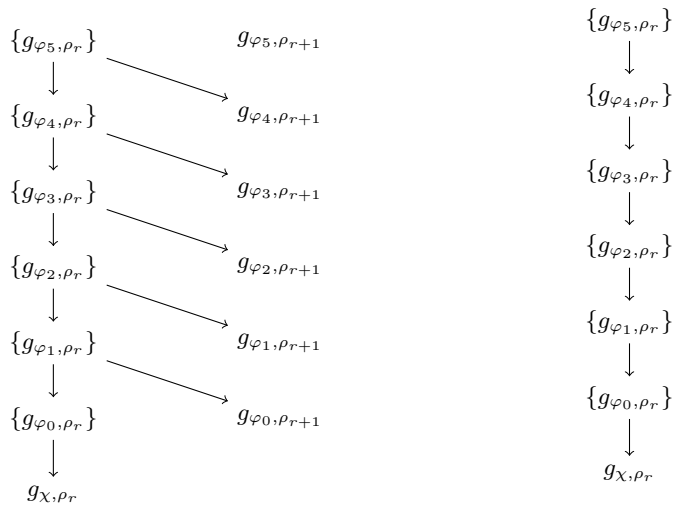


Figure 6.11: The circuits are equivalent to the circuits from Figure 6.10. Moreover, B remains an extended normal product of two paths after we substitute the corresponding extended gates in B by these circuits.

for $0 \leq r < |\rho|$. We finish the construction for $\chi \,\mathrm{U}_n\, \psi$ with the remark that the labeling β can by computed in L.

We will skip a detailed treatment of the remaining three bounded temporal connectives. We just mention that the case of R_n is dual to construction for U_n. The cases of S_n and T_n are completely analogous by reverting the order of the states in ρ.

Now we can use Theorem 12 to AC^1-reduce the evaluation of Γ to the evaluation of circuits with extended gates. Using the presented construction the evaluation of circuits with extended gates is reduced to normal OIF circuits with variable gates. The remaining details are the same as in the proof of Theorem 6 and Theorem 11. The overall complexity is $\mathsf{AC}^1(\mathsf{logDCFL})$. □

Chapter 7

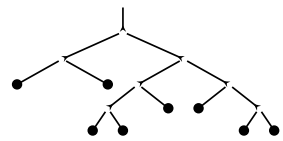
Conclusions

We have presented efficient parallel algorithms for checking LTL and CTL formulas over finite paths and trees. By the adaption of the construction to two important extensions of LTL the flexibility of our approach has been demonstrated. This suggests that the algorithmic concepts behind our construction may show useful in a broader context. The results are a significant step forward in the research program towards a complete picture of the complexities of the path checking problems across the spectrum of temporal logics, which was started in 2003 by Markey and Schnoebelen [71]. The main idea of our approach is the use of planar circuits as a representation of partially evaluated subformulas, which allows the evaluation of the formula to efficiently stop and resume, as dictated by the dependencies between the subformulas. We conjecture that the use of planar circuits as a data structure in parallel or space-efficient verification algorithms, following the pattern of our construction, will find applications in other model checking problems as well.

In this chapter we recall the results of the thesis along with some concluding remarks. We close the thesis by stating interesting open questions and promising directions of future research.

LTL path checking. We presented an $\mathsf{AC}^1(\mathsf{logDCFL})$ algorithm for checking LTL formulas over finite paths. This improves significantly over the previously best known upper bound of P.

Events characterized by finite languages are studied since Kleene's *definite events* [53] and the *locally testable events* of McNaughton and Papert [74]. In the terminology of McNaughton and Papert, a set E of words is called a *locally testable event in the strict sense* if there exists a finite language L, such that all subwords of each word in E have a prefix in L. McNaughton and Papert



construct an automaton that maintains an input buffer that is large enough to capture the largest words in L. In each step, a combinatorial circuit checks if the pipeline content belongs to L. In case that L is given as a temporal logic formula our results provide bounds on the depth of the combinatorial circuit.

CTL tree checking. We have shown that the tree checking problem for CTL is in $\mathsf{AC}^2(\mathsf{logDCFL})$.

As a consequence of our results, the best known upper bound for model checking CTL on trees, namely $\mathsf{AC}^2(\mathsf{logDCFL})$, is of higher complexity than the best known upper bound for model checking LTL on trees, namely $\mathsf{AC}^1(\mathsf{logDCFL})$. This is in contrast with the general model checking problem, which is PSPACE-complete for LTL and P-complete for CTL. We conjecture that on trees, CTL model checking is actually not easier than LTL model checking. The conjecture is based on the observation that for trees, the number of paths that an LTL formula has to be checked against is only linear (in contrast to general model checking where uncountably many paths have to be checked). Therefore, all paths can be checked individually in parallel without any blow-up. In contrast, for CTL the path quantors induce a hierarchical dependence between the suffixes of different paths in the structure. Whereas on general structures this allows for tractable model checking via a branch-and-bound approach, on trees it prevents a straightforward parallelization, since those dependencies must be resolved in a clever way. This comes at an extra price that might even result in a strictly higher complexity.

The tree checking problem is an important problem in the development of efficient algorithms for query languages for tree-shaped data-structures. We think that CTL provides an interesting alternative to the more expressive but expensive versions of XPath, such as Core XPath or $\mathcal{C}$XPath, on one side, and similarly cheap but very restrictive fragments like positive Core XPath on the other side. Compared to the latter CTL may be beneficial for queries that require universal quantification and the power of "until" but do not relay on the sibling axis.

Complexity of LTL model checking for classes of Kripke structures. We have developed a classification of Kripke structures with respect to the complexity of the model checking problem for LTL. We showed that the model checking problem for a Kripke structure is PSPACE-complete if and only if the Kripke structure is not weak. The problem is coNP-complete for the class of all weak Kripke structures. The problem is in NC for any class of Kripke structures for which the model checking problem can be reduced to a polynomial number of path checking problems.

Examples of such classes include finite paths, ultimately periodic path, finite trees, directed graphs of constant depths, and classes of Kripke structures with

a cycle graph of constant depth.

Path checking for extensions of LTL. We have shown that our approach to parallel path checking that is based on the evaluation of monotone Boolean circuits can be extended to LTL+Past and BLTL+Past.

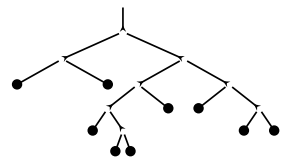
We have provided an $\mathsf{AC}^1(\mathsf{logDCFL})$ algorithm for checking BLTL+Past formulas over finite paths. While other extensions of LTL, for example with Chop or Past+Now, immediately render the path checking problem P-complete and, hence, inherently sequential [71], LTL with past and bounds can be checked efficiently in parallel.

There is a growing practical demand for efficient parallel algorithms, driven by the increasing availability of powerful (and inherently parallel) programmable hardware. For example, several tools are available that translate PSL assertions to hardware-based monitors [20, 15, 27]. Such implementations can immediately apply our construction to evaluate subformulas consisting of bounded and past operators in parallel rather than sequentially.

Unlike in static verification, where the verification algorithm is executed at design-time and can therefore afford to spend significant time and resources, runtime verification algorithms must run in synchrony with the monitored system and usually even share the resources of the implementation platform. Therefore, an online monitor must have a particularly low complexity foot-print for a single update-step. In [27] we provide a novel automata-based translation of temporal specifications to monitor circuits that saves exponential space in the size of bounded-future subformulas. The construction exploits the local testablility of bounded subformulas, that occur within general temporal properties, by the introduction of a pipeline into the monitoring circuit. Our result on BLTL+Past path checking allows to further improve the time complexity of an update-step by an exponential factor.

LTL+Past with only past-time modalities is a particularly attractive specification logic for online monitoring, since the space complexity is only linear in the formula and constant in the trace length. Because the update-step of the monitor can be reduced to the evaluation of a Boolean formula it is in NC^1. With our construction the speed of a pure past-time monitor can be further improved by buffering the input and evaluating chunks of n sequential observations efficiently in parallel.

Evaluation of monotone boolean circuits. Our algorithms for path checking and tree checking of temporal logic formulas rely on the efficient parallel evaluation of monotone Boolean circuits. In the thesis we show that for circuits that have a graph that is the normal graph products of trees and paths the evaluation problem can be reduced to the evaluation problem for monotone



planar Boolean circuits. These kinds of products of combinatorial structures are quite common. We expect that our techniques can be applied directly or can be extended to be useful in other application as well.

Open questions and future work. There are several open questions that deserve further attention. Albeit small, there is still a gap between $\mathsf{AC}^1(\mathsf{logDCFL})$ and the best known lower bound, NC^1 for LTL path checking. Similarly, there is a gap between $\mathsf{AC}^2(\mathsf{logDCFL})$ and NC^1 for CTL tree checking.

Hence, tight bounds for the complexity of LTL path checking remain a challenging open problem. There is some hope to further reduce the upper bound towards NC^1, the currently known lower bound, because our construction relies on the algorithms for evaluating monotone planar Boolean circuits with all constant gates on the outer face. The circuits that appear in our construction actually exhibit much more structure. However, we are not aware of any algorithm that takes advantage of that and performs better than logDCFL. Another way to improve the upper bounds of our path and tree checking algorithms would be to prove a better upper bound for the problem of evaluating one-input-face monotone planar Boolean circuits.

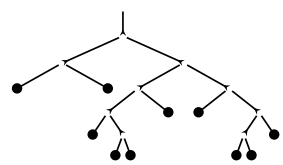
Beyond finite trees we do know very little about other classes for which the model checking problem for CTL is in NC. What are the properties of Kripke structures that allow for efficiently parallel model checking for CTL? What is the complexity of tree checking for $CTL + past$ and CTL with a sibling axis? What is the complexity of CTL^* tree checking? LTL and CTL both satisfy the basic properties that we require for our techniques to apply: They both have a linear positive normal form and expansion laws that correspond to normal graph products. CTL^* does not fulfill the second requirement. For example, the formula $A(Gp \vee Fq)$ can not be expanded in the required way.

An intriguing question is whether the path checking complexities of LTL and BLTL are actually the same: while they are both in NC, the circuits resulting from BLTL formulas seem to be combinatorially more complex.

The proofs in this thesis make use of different computational models: Boolean circuits, space-restricted Turing machines, time-restricted Turing machines, Turing machines with push-down store, and parallel random access memory machines (PRAM). Can we derive practical parallel implementations from our parallel path and tree checking algorithms?

Complexity classes that are characterized by efficient parallel algorithms and complexity classes that are characterized by space-efficient algorithms are tightly coupled through simulation theorems. In the light that in modern hardware architectures cache-efficiency and I/O-efficiency are the more important performance factors than the actual number of computation steps, the following question seems even more important than the previous one: Can we derive prac-

tical space-efficient implementations from our parallel path and tree checking algorithms? In particular with the fast growing number of available cores in modern computing devices, on the one hand, and the tight resource restrictions on mobile devices, on the other hand, good trade-offs between cache-efficiency, I/O-efficiency, and CPU-usage become more important.



Bibliography

[1] K. Abrahamson, N. Dadoun, D. Kirkpatrick, and T. Przytycka. A simple parallel tree contraction algorithm. *J. Algorithms*, 10(2):287–302, 1989.

[2] T. Ågotnes, W. van der Hoek, J. A. Rodríguez-Aguilar, C. Sierra, and M. Wooldridge. Multi-modal ctl: Completeness, complexity, and an application. *Studia Logica*, 92(1):1–26, 2009.

[3] R. Alur and T. A. Henzinger. A really temporal logic. In *FOCS*, pages 164–169. IEEE, 1989.

[4] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.

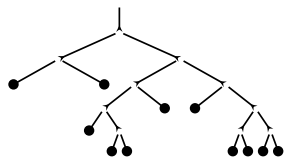
[5] R. Armoni, D. Korchemny, A. Tiemeyer, M. Vardi, and Y. Zbar. Deterministic dynamic monitors for linear-time assertions. In *FATES/RV'06*, LNCS. Springer, 2006.

[6] C. Artho, H. Barringer, A. Goldberg, K. Havelund, S. Khurshid, M. Lowry, C. Pasareanu, G. Rosu, K. Sen, W. Visser, and R. Washington. Combining test case generation and runtime verification. *Theoretical Computer Science*, 336(2-3):209 – 234, 2005.

[7] P. Barceló and L. Libkin. Temporal logics over unranked trees. In *LICS*, pages 31–40. IEEE Computer Society, 2005.

[8] D. Barrington, C.-J. Lu, P. Miltersen, and S. Skyum. On monotone planar circuits. In *COCO*, pages 24–31, Washington, DC, USA, 1999. IEEE Computer Society.

[9] A. Bauer, M. Leucker, and C. Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In O. Sokolsky and S. Tasiran, editors, *RV*, volume 4839 of *Lecture Notes in Computer Science*, pages 126 – 138. Springer, 2007.



[10] M. Bauland, M. Mundhenk, T. Schneider, H. Schnoor, I. Schnoor, and H. Vollmer. The tractability of model-checking for ltl: The good, the bad, and the ugly fragments. *Electr. Notes Theor. Comput. Sci.*, 231:277–292, 2009.

[11] M. Bauland, T. Schneider, H. Schnoor, I. Schnoor, and H. Vollmer. The complexity of generalized satisfiability for linear temporal logic. *Logical Methods in Computer Science*, 5(1), 2009.

[12] M. Benedikt, L. Libkin, and F. Neven. Logical definability and query languages over ranked and unranked trees. *ACM Trans. Comput. Log.*, 8(2), 2007.

[13] O. Bernholtz, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking (extended abstract). In D. L. Dill, editor, *CAV*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 1994.

[14] O. Beyersdorff, A. Meier, M. Thomas, H. Vollmer, M. Mundhenk, and T. Schneider. Model checking ctl is almost always inherently sequential. *Temporal Representation and Reasoning, International Syposium on*, 0:21–28, 2009.

[15] M. Boule and Z. Zilic. Automata-based assertion-checker synthesis of PSL properties. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 13(1), 2008.

[16] S. Buss. The boolean formula value problem is in ALOGTIME. In *STOC*, pages 123–131, New York, NY, USA, 1987. ACM.

[17] T. Chakraborty and S. Datta. One-input-face MPCVP is hard for L, but in LogDCFL. In *FSTTCS*, volume 4337 of *LNCS*, pages 57–68. Springer, 2006.

[18] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244 – 263, 1986.

[19] E. M. Clarke, S. Jha, R. Enders, and T. Filkorn. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.

[20] A. Dahan, D. Geist, L. Gluhovsky, D. Pidan, G. Shapir, Y. Wolfsthal, L. Benalycherif, R. Kamdem, and Y. Lahbib. Combining system level modeling with assertion based verification. In *ISQED'05*, pages 310–315. IEEE Computer Society, 2005.

[21] A. Delcher and S. Kosaraju. An NC algorithm for evaluating monotone planar circuits. *SIAM J. Comput.*, 24(2):369–375, 1995.

[22] S. Demri, F. Laroussinie, and P. Schnoebelen. A parametric analysis of the state-explosion problem in model checking. *J. Comput. Syst. Sci.*, 72(4):547–575, 2006.

[23] S. Demri and P. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Inf. Comput.*, 174(1):84–103, 2002.

[24] P. Dymond and S. Cook. Complexity theory of parallel time and hardware. *Information and Computation*, 80(3):205–226, 1989.

[25] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.

[26] E. A. Emerson and A. P. Sistla. Symmetry and model checking. In C. Courcoubetis, editor, *CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 463–478. Springer, 1993.

[27] B. Finkbeiner and L. Kuhtz. Monitor circuits for LTL with bounded and unbounded future. In *RV*, LNCS. Springer, 2009.

[28] B. Finkbeiner and H. Sipma. Checking finite traces using alternating automata. *Formal Methods in System Design*, 24:101–127, 2004.

[29] D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal basis of fairness. In *POPL*, pages 163–173, 1980.

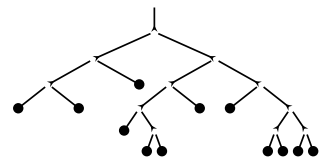
[30] M. Geilen. On the construction of monitors for temporal logic properties. *Electr. Notes Theor. Comput. Sci.*, 55(2), 2001.

[31] D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In *ASE*, pages 412 – 416. IEEE Computer Society, 2001.

[32] A. Gibbons and W. Rytter. *Efficient parallel algorithms*. Cambridge University Press, 1988.

[33] P. Godefroid and D. Pirottin. Refining dependencies improves partial-order verification methods (extended abstract). In *CAV*, pages 438–449, London, UK, 1993. Springer-Verlag.

[34] L. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–29, 1977.



[35] L. Goldschlager. A space efficient algorithm for the monotone planar circuit value problem. *Information Processing Letters*, 10(1):25–27, 1980.

[36] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. In *VLDB*, pages 95–106. Morgan Kaufmann, 2002.

[37] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.

[38] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of xpath query evaluation and xml typing. *J. ACM*, 52(2):284–335, 2005.

[39] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, 1995.

[40] K. Havelund and G. Roşu. Monitoring programs using rewriting. In *ASE*, pages 135 – 143. IEEE Computer Society, 2001.

[41] K. Havelund and G. Roşu. Efficient monitoring of safety properties. *STTT*, 2004.

[42] E. Hemaspaandra. The complexity of poor man's logic. *J. Log. Comput.*, 11(4):609–622, 2001.

[43] E. Hemaspaandra and H. Schnoor. On the complexity of elementary modal logics. In S. Albers and P. Weil, editors, *STACS*, volume 1 of *LIPIcs*, pages 349–360. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.

[44] E. Hemaspaandra, H. Schnoor, and I. Schnoor. Generalized modal satisfiability. *J. Comput. Syst. Sci.*, 76(7):561–578, 2010.

[45] IEEE Std 1850-2007. *Property Specification Language (PSL).* IEEE, New York, 2007.

[46] C. N. Ip and D. L. Dill. Better verification through symmetry. In D. Agnew, L. J. M. Claesen, and R. Camposano, editors, *CHDL*, volume A-32 of *IFIP Transactions*, pages 97–111. North-Holland, 1993.

[47] F. Jahanian and A. K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Trans. Software Eng.*, 12(9):890–904, 1986.

[48] T. Jiang and B. Ravikumar. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters*, 40:25–31, 1991.

[49] D. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 67–161. MIT Press, 1990.

[50] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California at Los Angeles (UCLA), 1968.

[51] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 869–942. MIT Press, 1990.

[52] S. Katz and D. Peled. An efficient verification method for parallel and distributed programs. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *REX Workshop*, volume 354 of *Lecture Notes in Computer Science*, pages 489–507. Springer, 1988.

[53] S. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*. Princeton University Press, 1956.

[54] S. Kosaraju. On parallel evaluation of classes of circuits. In *FSTTCS*, volume 472 of *LNCS*, pages 232–237. Springer, 1990.

[55] S. R. Kosaraju and A. L. Delcher. Optimal parallel evaluation of tree-structured computations by raking. In *VLSI Algorithms and Architectures: Proceedings of the 3rd Aegean Workshop on Computing*, pages 101–110. Springer-Verlag, 1988.

[56] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

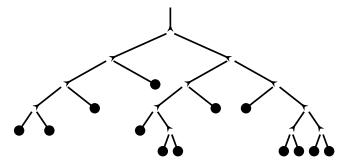
[57] L. Kuhtz. MoCS – Monitor Circuit Synthesis, 2009. `http://react.cs.uni-sb.de/tools/mocs`.

[58] L. Kuhtz and B. Finkbeiner. LTL path checking is efficiently parallelizable. In *ICALP'09*, volume 5556 of *LNCS*, pages 235 – 246. Springer, 2009.

[59] O. Kupferman and M. Vardi. Relating linear and branching model checking. In *PROCOMET*, pages 304–326, New York, June 1998. Chapman & Hall.

[60] A. Kučera and J. Strejček. The stuttering principle revisited. *Acta Inf.*, 41(7-8):415–434, 2005.

[61] R. E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975.



[62] R. E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6(3):467–480, 1977.

[63] F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *LICS*, pages 383–392. IEEE Computer Society, 2002.

[64] L. Libkin and C. Sirangelo. Reasoning about xml with temporal logics and automata. *J. Applied Logic*, 8(2):210–232, 2010.

[65] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL*, pages 97–107, 1985.

[66] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218, London, UK, 1985. Springer.

[67] N. Limaye, M. Mahajan, and J. Sarma. Evaluating monotone circuits on cylinders, planes and tori. In B. Durand and W. Thomas, editors, *STACS*, volume 3884 of *LNCS*, pages 660–671. Springer, 2006.

[68] N. Markey. Past is for free: on the complexity of verifying linear temporal properties with past. *Electr. Notes Theor. Comput. Sci.*, 68(2), 2002.

[69] N. Markey. Temporal logic with past is exponentially more succinct, concurrency column. *Bulletin of the EATCS*, 79:122–128, 2003.

[70] N. Markey and J.-F. Raskin. Model checking restricted sets of timed paths. *Theoretical Computer Science*, 358(2-3):273 – 292, 2006. Concurrency Theory (CONCUR 2004).

[71] N. Markey and P. Schnoebelen. Model checking a path (preliminary report). In *CONCUR*, volume 2761 of *LNCS*, pages 251–265. Springer, 2003.

[72] N. Markey and P. Schnoebelen. Mu-calculus path checking. *Inf. Process. Lett.*, 97(6):225–230, 2006.

[73] M. Marx. Conditional xpath. *ACM Trans. Database Syst.*, 30(4):929–959, 2005.

[74] R. McNaughton and S. Papert. *Counter-Free Automata*, volume 65 of *Research Monograph*. MIT Press, 1971.

[75] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[76] H. Petersen. Decision problems for generalized regular expressions. In *DCAGRS*, pages 22–29, 2000.

[77] H. Petersen. The membership problem for regular expressions with intersection is complete in LOGCFL. In *STACS*, volume 2285 of *LNCS*, pages 513–522. Springer, 2002.

[78] A. Pnueli. The temporal logic of programs. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:46–57, 1977.

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

[80] A. Prior. *Past, Present and Future.* Oxford University Press, 1967.

[81] V. Ramachandran and H. Yang. An efficient parallel algorithm for the layered planar monotone circuit value problem. In T. Lengauer, editor, *ESA*, volume 726 of *Lecture Notes in Computer Science*, pages 321–332. Springer, 1993.

[82] V. Ramachandran and H. Yang. An efficient parallel algorithm for the layered planar monotone circuit value problem. *Algorithmica*, 18(3):384–404, 1997.

[83] B.-H. Schlingloff. On the expressive power of modal logics on trees. In A. Nerode and M. A. Taitslin, editors, *LFCS*, volume 620 of *Lecture Notes in Computer Science*, pages 441–451. Springer, 1992.

[84] Ph. Schnoebelen. The complexity of temporal logic model checking. In Ph. Balbiani, N.-Y. Suzuki, F. Wolter, and M. Zakharyaschev, editors, *Selected Papers from the 4th Workshop on Advances in Modal Logics (AiML'02)*, pages 393–436, Toulouse, France, 2003. King's College Publication. Invited paper.

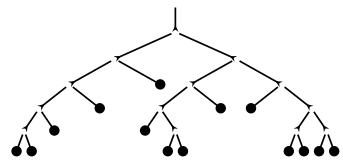
[85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[86] A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4):297–322, 1992.

[87] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer, 1999.

[88] World Wide Web Consortium. XQuery 1.0: A query language for XML. http://www.w3.org/TR/xquery/.

[89] World Wide Web Consortium. XSL transformations language (XSLT): Version 2.0. http://www.w3.org/TR/xslt20/.



[90] World Wide Web Consortium. XML path language (XPath): Version 1.0, 1999. http://www.w3c.org/TR/xpath.

[91] H. Yang. An NC algorithm for the general planar monotone circuit value problem. In *IPDPS*, pages 196–203, 1991.

[92] H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV*, volume 2404 of *LNCS*. Springer, 2002.