Logical Methods in Computer Science
Vol. 8(4:10)2012, pp. 1–24
www.lmcs-online.org

Submitted Jan. 4, 2010
Published Oct. 23, 2012

EFFICIENT PARALLEL PATH CHECKING FOR LINEAR-TIME TEMPORAL LOGIC WITH PAST AND BOUNDS *

LARS KUHTZ AND BERND FINKBEINER

Universität des Saarlandes, 66123 Saarbrücken, Germany
e-mail address: {kuhtz, finkbeiner}@cs.uni-saarland.de

Abstract. Path checking, the special case of the model checking problem where the model under consideration is a single path, plays an important role in monitoring, testing, and verification. We prove that for linear-time temporal logic (LTL), path checking can be efficiently parallelized. In addition to the core logic, we consider the extensions of LTL with bounded-future (BLTL) and past-time (LTL+Past) operators. Even though both extensions improve the succinctness of the logic exponentially, path checking remains efficiently parallelizable: Our algorithm for LTL, LTL+Past, and BLTL+Past is in $\mathsf{AC}^1(\mathsf{logDCFL}) \subseteq \mathsf{NC}$.

1. Introduction

Linear-time temporal logic (LTL) is the standard specification language to describe properties of computation paths. The problem of checking whether a given finite path satisfies an LTL formula plays a key role in monitoring and runtime verification [14, 12, 7, 2, 5], 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 [3] and in several static verification techniques, notably in Monte-Carlo-based probabilistic verification, where large numbers of randomly generated sample paths are analyzed [31].

Somewhat surprisingly, given the widespread use of LTL, the complexity of the path checking problem is still open [26]. The established upper bound is P: The algorithms in the literature traverse the path sequentially (cf. [12, 26, 14]); 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 running time that is quadratic in the size of the formula plus the length of the path. The only known lower bound is NC^1 [9], the complexity of evaluating

1998 ACM Subject Classification: F.4.1, F.2.2.

Key words and phrases: linear-time temporal logic (LTL), linear-time temporal logic with past, bounded temporal operators, model checking, path checking, parallel complexity.

* Theorem 1 of this paper has been published before in [22]. The results in this paper are also included in the Ph.D. thesis of the first author [21].

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

DOI:10.2168/LMCS-8(4:10)2012

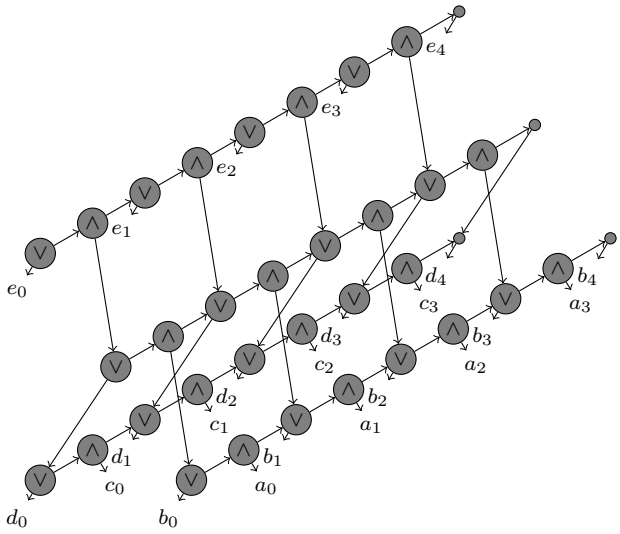

Figure 1: Circuit resulting from unrolling the LTL formula $((a \,\mathrm{U}\, b) \,\mathrm{U}\, (c \,\mathrm{U}\, d)) \,\mathrm{U}\, e$ over a path ρ of length 5. We denote the value of an atomic proposition p at a path position $i = 0, \ldots, 4$ by p_i.

Boolean expressions. 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 temporal logic PSL [15], an extension of LTL, has become part of the hardware description language VHDL, and several tools [7, 5, 11] 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?

Parallelizing LTL path checking. We show that LTL path checking can indeed be parallelized efficiently. Our approach is inspired by work in the related area of evaluating monotone Boolean circuits [13, 10, 20, 4, 25, 6]. Rather than sequentially traversing the path, we consider the circuit that results from unrolling the formula in positive normal form over the path using the expansion laws of the logic. Using the positive normal form of the formula ensures that the resulting circuit is monotone. The size of the circuit is quadratic in the size of formula plus the size of the path. For logarithmic measures, the circuit thus is of the same order as the input. Figure 1 shows such a circuit for the formula $((a \,\mathrm{U}\, b) \,\mathrm{U}\, (c \,\mathrm{U}\, d)) \,\mathrm{U}\, e$ and a path of length 5.

Yang [30] and, independently, Delcher and Kosaraju [8] have shown that monotone Boolean circuits can be evaluated efficiently in parallel *if the graph of the circuit has a planar embedding*. Unfortunately, this condition is already violated in the simple example of Figure 1 as shown in Figure 2. Individually, however, each operator results in a planar circuit: for example, $d \,\mathrm{U}\, e$ results in $e_0 \vee (d_0 \wedge (e_1 \vee (d_1 \wedge \ldots) \cdots)$. The complete formula thus defines a tree of planar circuits.

Our path checking algorithm works on this tree of circuits. We perform a parallel tree contraction [1, 19, 18] to collapse a parent node and its children nodes into a single planar circuit. Simple paths in the tree immediately collapse into a planar circuit; the remaining

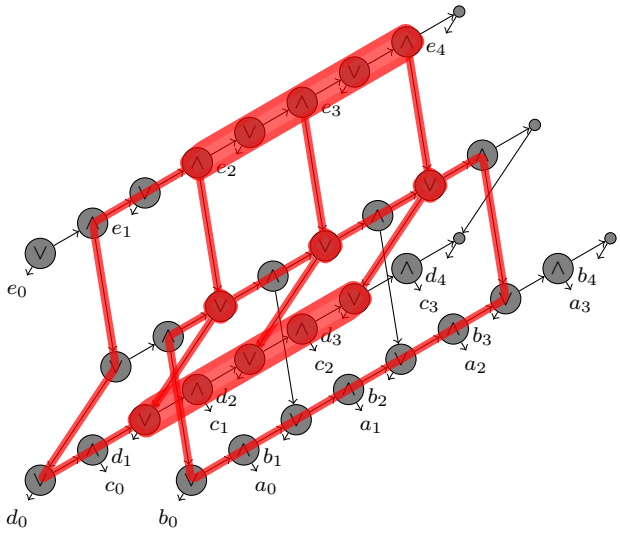

Figure 2: Circuit resulting from unrolling the LTL formula $((a \,\mathrm{U}\, b) \,\mathrm{U}\, (c \,\mathrm{U}\, d)) \,\mathrm{U}\, e$ over a path ρ of length 5. The red colored minor of the graph of the circuit is a K_5. Thus the circuit is not planar.

binary tree is contracted incrementally, until only a single planar circuit remains. The key insight for this construction is that a contraction can be carried out *as soon as one of the children has been evaluated*. Initially, all leaves correspond to atomic propositions. During the contraction all leaves are evaluated. Because no leaf has to wait for the evaluation of its sibling before it can be contracted with its parent, we can contract a fixed portion of the nodes in every sequential step, and therefore terminate in at most a logarithmic number of steps.

The path checking problem can, hence, be parallelized efficiently. The key properties of LTL that are exploited in the construction are the existence of a positive normal form of linear size and expansion laws that, when iteratively applied, increase the size of the Boolean circuit only linearly in the number of iteration steps. The combinatorial structure of the resulting circuit allows for an efficient reduction of the evaluation problem to the evaluation problem of planar Boolean circuits. In addition to planarity, our construction maintains some further technical invariants, in particular that the circuits have all input gates on the outer face. Analyzing this construction, we obtain the result that the path checking problem is in $\mathsf{AC}^1(\mathsf{logDCFL})$:

Theorem 1. *The LTL path checking problem is in* $\mathsf{AC}^1(\mathsf{logDCFL})$.

The $\mathsf{AC}^1(\mathsf{logDCFL})$ complexity results from a reduction that is performed by an outer algorithm, which can be implemented as a uniform family of Boolean circuits of logarithmic depth, and an inner operation, which is represented through unbounded fan-in gates that are embedded in the circuits and that serve as $\mathsf{logDCFL}$ oracles. The AC^1 complexity of the outer reduction is due to the tree contraction algorithm. Throughout the contraction, each atomic contraction operation processes a sub-circuit with a size that is of the same order as the overall input circuit. Hence, the oracle gates have non-constant fan-in. The whole tree contraction is performed in a logarithmic number of parallel steps. Thus, the contraction

algorithm implements an AC^1 reduction. Within the AC^1 contraction circuits, the oracle gates perform the evaluation of a certain class of monotone planar Boolean circuits. This operation can be bound to a complexity of $\mathsf{logDCFL}$. In summary, the overall path checking algorithm consists of two sequential reduction steps: First the LTL path checking problem is reduced (in logarithmic space) to the problem of evaluating a certain class of monotone Boolean circuits. Second, the problem of evaluating those circuits is AC^1 reduced to the problem of evaluating a certain class of monotone planar Boolean circuits. The latter problem is solved by an $\mathsf{logDCFL}$ oracle.

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 [16], in logCFL for semi-extended regular expressions [28], and P-complete for star-free regular expressions and extended regular expressions [27]. Of particular interest is the comparison to the star-free regular expressions, since they have the same expressive power as LTL on finite paths [24]. With $\mathsf{AC}^1(\mathsf{logDCFL})$ vs. P, our result demonstrates a computational advantage for LTL.

LTL with past and bounded-future operators. Practical temporal logics like PSL extend LTL 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 [27]. 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 Schoebelen [23] 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. However, since past operators are the dual of future operators, they also result in planar circuits; hence, the construction for LTL can directly be applied to the tree of circuits that results from LTL formulas with unbounded past and future operators and we obtain the following result:

Theorem 2. *The LTL+Past path checking problem is in* $\mathsf{AC}^1(\mathsf{logDCFL})$.

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 [11], 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 unfold the formula to the core fragment and then apply the construction described above

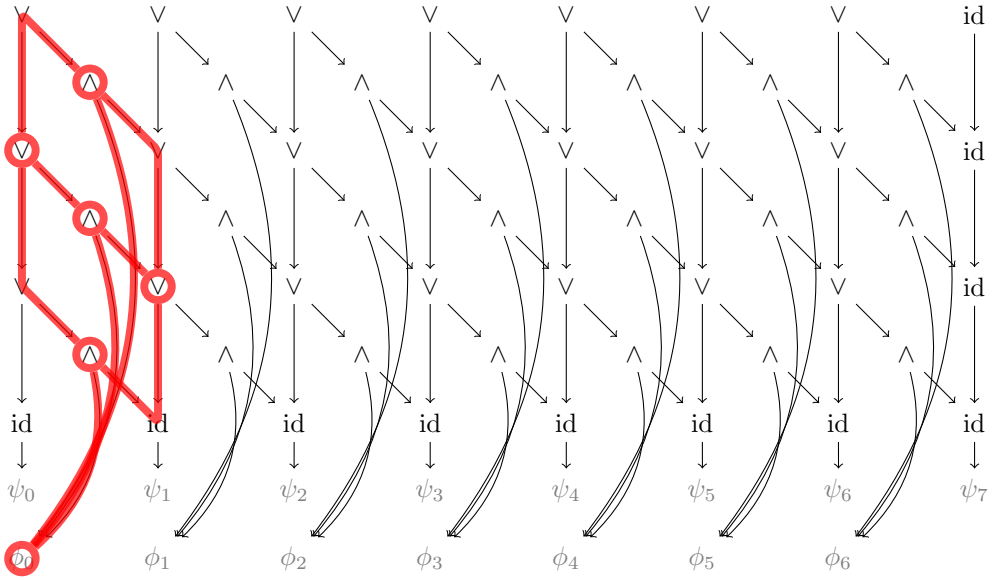

Figure 3: The circuit for the bounded formula $\phi \,\mathrm{U}_3\, \psi$. Since the red colored subgraph is a $K_{3,3}$, the circuit has no planar embedding. However, if the ϕ_i-gates are constants, then propagating the constants eliminates the edges that prevent the shown embedding from being planar.

for the LTL operators. Because of the doubly exponential blow-up, however, such a solution would no longer be in NC. If we instead apply the expansion laws for the bounded operators to the original formula, we obtain a circuit of polynomial size, but with a more complex structure. Because of this more complex structure, the path checking construction that we described above for the core logic is no longer applicable. Consider the circuit corresponding to the bounded formula $\phi \,\mathrm{U}_3\, \psi$, shown in Figure 3: Since the graph of the circuit contains a $K_{3,3}$ subgraph, it has no planar embedding. Translating a formula with bounded operators to a tree of circuits would thus include non-planar circuits, which in general cannot be evaluated efficiently in parallel.

The key insight of the construction for the extended logic is that, although the circuit for the bounded operators is not planar *a priori*, an *equivalent* planar circuit can be constructed *as soon as one of the direct subformulas has been evaluated.* Suppose, for example, that the ϕ_i-gates in the circuit shown in Figure 3 are constants. Propagating these constants *eliminates* all edges that prevent the shown embedding from being planar! In general, simple propagation is not enough to make the circuit planar. This is illustrated in Figure 4, where the same formula is analyzed under the assumption that the ψ_i-gates are constant. While the propagation of the constants replaces parts of the circuit (identified by the dotted lines) with constants, there remain references to ϕ_i-gates, e.g., the two references to ϕ_2, that prevent the shown embedding from being planar. However, an equivalent planar circuit exists: This circuit, shown in Figure 4 as a gray overlay, replaces the disturbing references to the ϕ_i-gates by vertical edges to subcircuits. For example, the first occurrence of ϕ_2 in $\phi_2 \wedge (0 \vee (\phi_2 \wedge (0 \vee (\psi_3 \wedge 1))$ is replaced with an edge to the subcircuit $\phi_2 \wedge (0 \vee (\psi_3 \wedge 1))$. The resulting circuit is equivalent, because the additional conjunct is redundant.

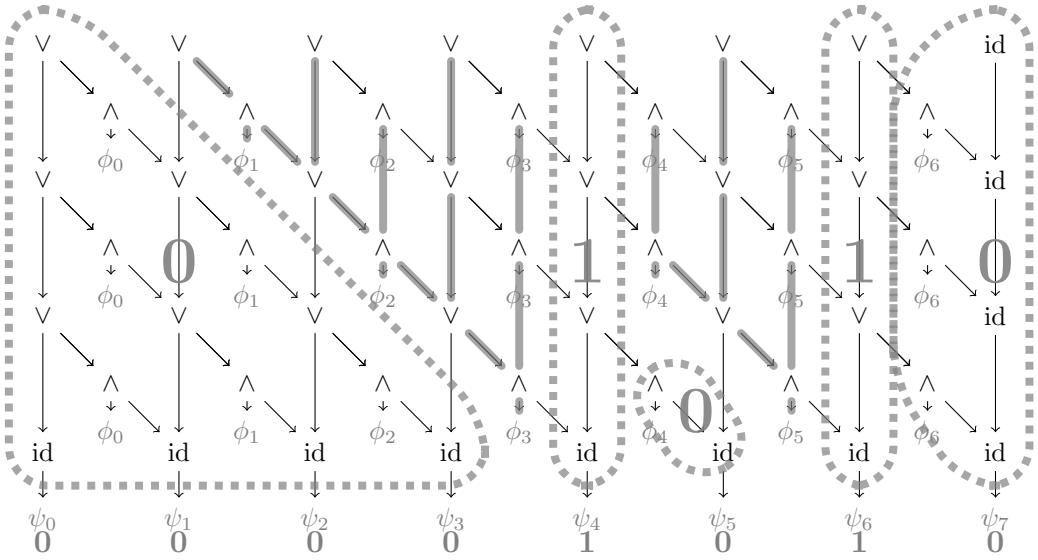

Figure 4: The circuit for the bounded formula $\phi \,\mathrm{U}_3\, \psi$ from Figure 3. If the ψ_i-gates evaluate to the constants shown in the bottom line, then the circuit depicted as a gray-colored overlay is an equivalent planar circuit.

Based on these observations, we present a translation from bounded temporal formulas to circuits that is guaranteed to produce planar circuits, but requires that one of the direct subformulas has already been evaluated. To meet this requirement, our path checking algorithm generates the circuits *on-the-fly*: a circuit for a subformula ϕ is constructed only when a direct subformula of ϕ is already evaluated. In this way, we avoid the construction of circuits that cannot be evaluated efficiently in parallel. As in the algorithm for LTL, we evaluate a fixed portion of the subformulas in every sequential step and thus terminate in time logarithmic in the size of the formula (bounds are encoded in $O(1)$) plus the length of the path. We prove the following result:

Theorem 3. *The BLTL+Past path checking problem is in* $\mathsf{AC}^1(\mathsf{logDCFL})$.

The remainder of the paper is structured as follows. After preliminaries in Section 2, Section 3 discusses the evaluation of monotone Boolean circuits. We introduce *transducer circuits*, which are circuits with a defined interface of input and output gates, and show that the composition of two planar transducer circuits can be computed in logDCFL. In Section 4, we describe the on-the-fly translation of BLTL+Past-formulas to planar circuits. In Section 5, we present the parallel path checking algorithm. We conclude with pointers to open questions in Section 6.

2. Preliminaries

2.1. **Linear-Time Temporal Logic.** 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 [24]. Let P be a set of atomic propositions. The *LTL formulas* are defined

inductively as follows: every atomic proposition $p \in P$ is a formula. If ϕ and ψ are formulas, then so are

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

The size of a formula ϕ is denoted by $\|\phi\|$.

LTL formulas are evaluated over computation paths. A *path* $\rho = \rho_0, \ldots, \rho_{n-1}$ is a finite sequence of states where each *state* ρ_i for $i = 0, \ldots, n-1$ is a valuation $\rho_i \in 2^P$ of the atomic propositions. The *length* of ρ is n and is denoted by $\|\rho\|$. Given an LTL formula ϕ, a nonempty path ρ satisfies ϕ at position i ($0 \leq i < \|\rho\|$), denoted by $(\rho, i) \models \phi$, if one of the following holds:

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

An LTL formula ϕ is *satisfied* by a nonempty path ρ (denoted by $\rho \models \phi$) iff $(\rho, 0) \models \phi$. By $\phi(\rho)$ we denote the Boolean sequence $s \in \mathbb{B}^{\|\rho\|}$ with $s_i = 1$ if and only if $(\rho, i) \models \phi$ for $0 \leq i < \|\rho\|$.

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:

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

We now extend LTL with the *past-time operators* $\mathrm{Y}^{\exists}$ (strong Yesterday), $\mathrm{Y}^{\forall}$ (weak Yesterday), S (Since), and T (Trigger) with the following semantics:

- $(\rho, i) \models \mathrm{Y}^{\exists}\psi$ iff
$$i - 1 \geq 0 \text{ and } (\rho, i-1) \models \psi\ ,$$
- $(\rho, i) \models \mathrm{Y}^{\forall}\psi$ iff
$$i - 1 < 0 \text{ or } (\rho, i-1) \models \psi\ ,$$
- $(\rho, i) \models \phi_l\,\mathrm{S}\,\phi_r$ iff
$$\exists j, i \geq j \geq 0.(\rho, j) \models \phi_r \text{ and } \forall k, i \geq k > j.(\rho, k) \models \phi_l\ ,\ \text{and}$$

- $(\rho, i) \models \phi_l \,\mathrm{T}\, \phi_r$ iff
$$\forall j, i \geq j \geq 0.(\rho, j) \models \phi_r \text{ or } \exists k, i \geq k > j.(\rho, k) \models \phi_l \ .$$

We call the resulting logic *linear-time temporal logic with past (LTL+Past)*. The following dualities ensure that each LTL+Past formula ϕ can be rewritten into a formula ϕ' in positive normal form with $\|\phi'\| = O(\|\phi\|)$.

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

The expansion laws for the past operators are

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

To obtain *linear-time temporal logic with past and bounds (BLTL+Past)* 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, the size of a formula is defined using unary encoding for the bounds. However, our results are actually indepent of the 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
$$\exists j, i \leq j \leq \min(i + b, \|\rho\| - 1).(\rho, j) \models \phi_r \text{ and } \forall k, i \leq k < j.(\rho, k) \models \phi_l \ ,$$
- $(\rho, i) \models \phi_l \,\mathrm{R}_b\, \phi_r$ iff
$$\forall j, i \leq j \leq \min(i + b, \|\rho\| - 1).(\rho, j) \models \phi_r \text{ or } \exists k, i \leq k < j.(\rho, k) \models \phi_l \ ,$$
- $(\rho, i) \models \phi_l \,\mathrm{S}_b\, \phi_r$ iff
$$\exists j, i \geq j \geq \max(i - b, 0).(\rho, j) \models \phi_r \text{ and } \forall k, i \geq k > j.(\rho, k) \models \phi_l \ , \text{ and}$$
- $(\rho, i) \models \phi_l \,\mathrm{T}_b\, \phi_r$ iff
$$\forall j, i \geq j \geq \max(i - b, 0).(\rho, j) \models \phi_r \text{ or } \exists k, i \geq k > j.(\rho, k) \models \phi_l \ .$$

The following dualities apply to the BLTL+Past operators:

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

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

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

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

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

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

We are interested in determining if a formula is satisfied by a given path. This is the path checking problem.

Definition 2.1 (Path Checking Problem)**.** The path checking problem for LTL (LTL+Past, BLTL+Past) is to decide, for an LTL (LTL+Past, BLTL+Past) formula ϕ and a nonempty path ρ, whether $\rho \models \phi$.

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

Lemma 2.2. *Given a BLTL+Past formula ϕ and a finite computation path ρ, the BLTL+Past 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. □

2.2. **Complexity classes within P.** We assume familiarity with the standard complexity classes within P. NC is the class of decision problems decidable in polylogarithmic time on a parallel computer with a polynomial number of processors. L is the class of problems that can be decided by a logspace restricted deterministic Turing machine. $\mathsf{logDCFL}$ 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 stack. $\mathsf{AC}^i, i \in \mathbb{N}$, denotes the class of problems decidable by polynomial size unbounded fan-in Boolean circuits of polylogarithmic depth of degree i. AC is defined as $\bigcup_{i\in\mathbb{N}} \mathsf{AC}^i$. Throughout the paper, all circuits are assumed to be uniform. Often we use functional versions of complexity classes. Since in our case the output size of the 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. Thus, in the following we do not explicitly distinguish between decision problems and functional problems [17]. It holds that

$$\mathsf{AC}^0 \subsetneq \mathsf{L} \subseteq \mathsf{logDCFL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{AC}^2 \subseteq \cdots \subseteq \mathsf{AC} = \mathsf{NC} \subseteq \mathsf{P} \ .$$

Further details can be found in the survey paper by Johnson [17].

Given a problem P and a complexity class C, P is AC^1 Turing reducible to C (denoted as $P \in \mathsf{AC}^1(C)$) if there is a family of AC^1 circuits with additional unbounded fan-in C-oracle gates that decides P. It holds that

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

For further details on AC^1 reductions, we refer to [29].

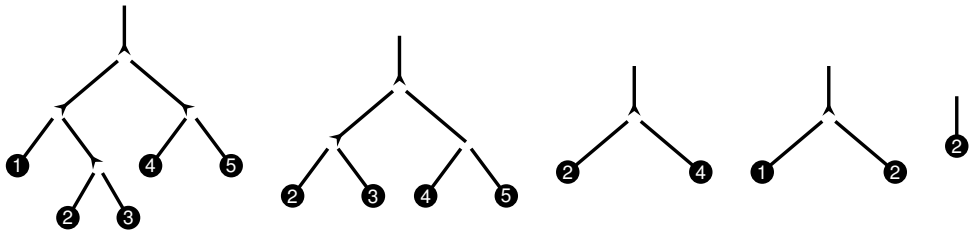

Figure 5: An parallel contraction process as produced by Algorithm 2.3.

2.3. **Parallel Tree Contraction.** The path checking algorithm presented in this paper relies on efficient parallel tree contraction. Here we follow the approach of [1] and [19]. A rooted binary tree is called *regular* if all inner nodes have exactly two children. Let $\mathcal{T}_0 = \langle V_0, E_0 \rangle$ be an ordered, rooted, regular, binary tree. A contraction step on $\mathcal{T}_i$ 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

$$V_{i+1} = (V_i \setminus \{l, p\}), \text{ and}$$

$$E_{i+1} = \begin{cases} E_i \setminus \{\langle p, l\rangle, \langle p, s\rangle\} & \text{if } p \text{ is the root of } \mathcal{T}_i, \\ (E_i \setminus \{\langle p, l\rangle, \langle p, s\rangle, \langle q, p\rangle\}) \cup \{\langle q, s\rangle\}, q \in V_i \text{ is parent of } p & \text{otherwise.} \end{cases}$$

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 an ordered, rooted, 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 2.3 from [18] performs a tree contraction in $\lceil \log n \rceil$ stages of parallel contraction steps.

Algorithm 2.3.
Input: an ordered, rooted, 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.
Update the numbering of the remaining (even numbered) leaves by dividing each leaf number by two.
end for

The algorithm can be implemented on an *exclusive read exclusive write random access memory machine (EREW PRAM)* such that it runs in time $O(\log n)$ with a total work of $O(n)$ [18]. 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 [29]. Figure 5 shows a tree contraction process for an example tree.

In order to use the parallel tree contraction algorithm to compute some function f on a labeled tree, the contraction step is piggybacked with a local operation on the labels of the nodes involved in the contraction step. In order for f to be in AC^1, the individual

contraction steps must be performed in constant time. For our constructions this is not the case. However, by piggybacking the contraction step with operations that for some complexity class $\mathcal{C}$ are solvable with $\mathcal{C}$-oracle gates, the problem of computing f is AC^1-reduced to $\mathcal{C}$. Hence, by showing that the complexity of the contraction step is in $\mathcal{C}$, the overall complexity of f is proven to be in $\mathsf{AC}^1(\mathcal{C})$.

3. Monotone Boolean Circuits

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 either with a Boolean value, with the symbol ?, with a tuple $\langle op, left, right \rangle$, or with a tuple $\langle id, suc \rangle$, where $op \in \{and, or\}$, and *left*, *right*, and *suc* are gates.

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 non-constant, non-variable gate a labeled with $\langle op, b, c \rangle$ or $\langle id, b \rangle$, we say that *a directly depends* on b and c, denoted by $a \succ b$, $a \succ c$. The *dependence* relation 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.*

For a circuit $G = \langle \Gamma, \gamma \rangle$, $\mathrm{const}(G)$ denotes the set of all constant gates in Γ. If $\Gamma = \mathrm{const}(G)$, we call G *constant*. By $\mathrm{var}(G)$ the set of all variable gates of Γ is denoted. Finally we define $\mathrm{src}(G)$ 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.

3.1. Circuit evaluation.

The *evaluation* of a circuit $\langle \Gamma, \gamma \rangle$ is the (unique) circuit $\langle \Gamma, \gamma' \rangle$ where for each gate $g \in \Gamma$ the following holds:

- $\gamma'(g) = 0$ iff $\gamma(g) = \langle and, l, r \rangle$ and $\gamma'(l) = 0$ or $\gamma'(r) = 0$,
- $\gamma'(g) = 1$ iff $\gamma(g) = \langle and, l, r \rangle$ and $\gamma'(l) = 1$ and $\gamma'(r) = 1$,
- $\gamma'(g) = \langle id, l \rangle$ iff $\gamma(g) = \langle and, l, r \rangle$ and $\gamma'(l) \notin \{0, 1\}$ and $\gamma'(r) = 1$,
- $\gamma'(g) = \langle id, r \rangle$ iff $\gamma(g) = \langle and, l, r \rangle$ and $\gamma'(r) \notin \{0, 1\}$ and $\gamma'(l) = 1$,
- $\gamma'(g) = 0$ iff $\gamma(g) = \langle or, l, r \rangle$ and $\gamma'(l) = 0$ and $\gamma'(r) = 0$,
- $\gamma'(g) = 1$ iff $\gamma(g) = \langle or, l, r \rangle$ and $\gamma'(l) = 1$ or $\gamma'(r) = 1$,
- $\gamma'(g) = \langle id, l \rangle$ iff $\gamma(g) = \langle or, l, r \rangle$ and $\gamma'(l) \notin \{0, 1\}$ and $\gamma'(r) = 0$,
- $\gamma'(g) = \langle id, r \rangle$ iff $\gamma(g) = \langle or, l, r \rangle$ and $\gamma'(r) \notin \{0, 1\}$ and $\gamma'(l) = 0$,
- $\gamma'(g) = \gamma'(s)$ iff $\gamma(g) = \langle id, s \rangle$ and $\gamma'(s) \in \{0, 1\}$, and
- $\gamma'(g) = \gamma(g)$ otherwise.

A circuit is *evaluated* if all constant gates are sink gates. 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: some *and*-/*or*-/*id*-gates are labeled as constant or *id*-gates.

The problem of evaluating monotone planar circuits has been studied extensively in the literature [13, 10, 20, 4, 25, 6]. Our construction is based on the evaluation of one-input-face planar circuits: Given a circuit $G = \langle \Gamma, \gamma \rangle$ with variable gates X, the *graph* $\mathrm{gr}(G)$ *of* G is the directed graph $\langle \Gamma, E \rangle$, where $E = \{\langle a, b \rangle \in \Gamma \times \Gamma \mid a \succ b\}$. A circuit C is *planar* if there exists a planar embedding of the graph of C. A planar circuit G is *one-input-face* if there is a planar embedding such that all gates of $\mathrm{src}(G)$ are located on the outer face. Note that

an evaluated planar circuit with all variable gates on the outer face is one-input-face. The evaluation of one-input-face planar circuits can be parallelized efficiently. We make use of a result by Chakraborty and Datta [6]:

Theorem 3.1 (Chakraborty and Datta 2006)**.** *The problem of evaluating an one-input-face planar circuit without variable gates is in* logDCFL.

Using standard techniques [20], the theorem generalizes to circuits that contain variable gates:

Corollary 3.2. *The problem of evaluating an one-input-face planar circuit 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. □

3.2. **Transducer Circuits.** The central construction in our path checking algorithm is circuit composition: circuits for larger subformulas are built from circuits for smaller subformulas by connecting variable gates of one circuit to gates of another circuit. To facilitate this operation, we introduce transducer circuits, which are circuits with a defined interface of input and output gates that allow the circuit to transform a sequence of Boolean input values, for example the values of a subformula at different positions of the path, into a sequence of output values.

A *transducer circuit* is a tuple $T = \langle \Gamma, \gamma, I, O\rangle$ where $G = \langle \Gamma, \gamma\rangle$ is a circuit, I is a (strict) ordering of $\mathrm{var}(G)$, and O is a (strict) ordering of a subset of Γ. I is called the input of T and O is called the output of T. The *input* and *output arity* is the length of the input and output, denoted as $\|I\|$ and $\|O\|$, respectively. We denote the i^{th} element of I and O by $I(i)$ and $O(i)$, respectively. The transducer circuit T is *planar* if G has a planar embedding such that the gates of I appear counter-clockwise ordered on the outer face, the gates of O appear clockwise ordered on the outer face, and between any two gates of I on the outer face there are either no or all gates of O, i.e., the gates of I and O do not appear interleaved on the outer face.

Given two planar transducer circuits $G = \langle \Gamma, \gamma, I_G, O_G\rangle$ and $D = \langle \Delta, \delta, I_D, O_D\rangle$, G is *composable with* D if the input arity of D equals the output arity of G. The *composition* $G \circ D$ of G with D is the planar transducer circuit $E = \langle \mathsf{E}, \epsilon, I_E, O_E\rangle$ with $\mathsf{E} = \Gamma \dot\cup \Delta$, $I_E = I_G$, $O_E = O_D$ and

$$\epsilon(g) = \begin{cases} \gamma(g), & \text{for } g \in \Gamma, \\ \delta(g), & \text{for } g \in \Delta \setminus \mathrm{var}(\Delta), \text{ and} \end{cases}$$
$$\epsilon(I_D(i)) = \langle id, O_G(i)\rangle, \text{ for } 0 \le i < \|O_G\|.$$

The composition $G \circ D$ can be computed by a logspace restricted deterministic Turing machine.

A transducer circuit T represents a function $f_T : \mathbb{B}^{\|I\|} \to \mathbb{B}^{\|O\|}$, where $f_T(s)$ for some sequence $s \in \mathbb{B}^{\|I\|}$ is computed by evaluating the composition of T with the constant circuit that represents s. The values of the output gates of the resulting constant circuit define the sequence $f_T(s)$.

Lemma 3.3. *For two one-input-face planar transducer circuits G and D, such that G is composable with D, the evaluation of $G \circ D$ is an evaluated one-input-face planar transducer circuit and can be computed within* logDCFL.

Proof. Let $G = \langle \Gamma, \gamma, I_G, O_G \rangle$ and $D = \langle \Delta, \delta, I_D, O_D \rangle$. The transducer circuits G' and D' are obtained from G and D, respectively, as follows. For each i, $0 \leq i < \|O_G\|$, with $O_G(i)$ being a constant gate b in G

- b is removed from G', including $O'_G(i)$,
- b is added to D',
- $\delta'(I_D(i)) = b$, and
- $I_D(i)$ is removed from I'_D.

In other words: all constant outputs of G are moved out of G' into D'. Clearly, G' is composable with D' and the evaluation of $G' \circ D'$ equals the evaluation of $G \circ D$. Further, G' and D' are both one-input-face and planar. Because G' is one-input-face, all constants are either on the outer face or are sinks. Since all constant gates that are sinks in G but not in $G \circ D$ have been moved out of G' into D' it holds that all constants in G' are on the outer face or are sinks also in $G' \circ D'$. $G' \circ D'$ generally is not one-input-face, because $\mathrm{src}(D')$ can contain constant gates that are neither sinks nor on the outer face of $G' \circ D'$, thus preventing the application of Theorem 3.1. However, D' is one-input-face and planar and can thus be evaluated in logDCFL using Theorem 3.1, resulting in a circuit D'' where all constants are sinks. Now, in the composition of G' with D'' all constants are either on the outer face of G' or are sinks. Thus, $G' \circ D''$ is an one-input-face planar transducer circuit that can be evaluated in logDCFL. The circuits G', and D', as well as the circuit compositions are computable in logarithmic space. □

4. Constructing Circuits On-The-Fly

We now describe the translation of BLTL+Past-formulas in positive normal form to planar circuits. As discussed in the introduction, the translation is not done as a preprocessing step, but rather delayed until *one* of the direct subformulas has been evaluated. We guarantee that the resulting circuit is planar, one-input-face, and evaluated. The path checking algorithm, which will be presented in the next section, composes the evaluated one-input-face planar circuits in order to represent larger partially evaluated subformulas.

Given a path ρ and a BLTL+Past formula ϕ in positive normal form with at most one unevaluated direct subformula, the following construction provides a function cir_ρ that maps the top-level operator of ϕ and its evaluated subformulas to an evaluated one-input-face planar transducer circuit that represents a partial evaluation of ϕ on ρ. The output arity of the circuit is $\|\rho\|$, the input arity is $\|\rho\|$ for all formulas except for atomic propositions, where the circuit has input arity 0. The circuit can be constructed by a logspace restricted Turing machine. The full details of the construction are provided in the appendix.

4.1. Atomic propositions.

For an atomic proposition p, the circuit is a set of constant gates, one for each path position. The value of a gate is the value of p at the respective position of ρ: $\mathrm{cir}_\rho(p) = \langle \{o_0, \ldots, o_{n-1}\}, l, \varepsilon, O \rangle$, where $n = \|\rho\|$, $O = o_0, \ldots, o_{n-1}$, $l(o_i) = 1$ iff $p \in \rho_i$, and ε denotes an empty input sequence. Clearly, a set of constant gates is an evaluated one-input-face planar transducer circuit.

4.2. **Unary operators.** For the unary operators $\mathrm{X}^\exists, \mathrm{X}^\forall, \mathrm{Y}^\exists$, and $\mathrm{Y}^\forall$, the circuit shifts the value of the input by one position in the respective direction. The first (respective last) position of the output is a constant with value 0 for strong operators and value 1 for weak operators. Again, the circuits are obviously planar, one-input-face, and evaluated, and of input and output arity $\|\rho\|$. E.g. $\mathrm{cir}_\rho(\mathrm{X}^\exists) = \langle G, l, I, O\rangle$, where $n = \|\rho\|$, $I = v_0, \ldots, v_{n-1}$, $O = o_0, \ldots, o_{n-1}$, $G = \{v_0, \ldots, v_{n-1}\} \cup \{o_0, \ldots, o_{n-1}\}$, and

$$l(o_i) = \begin{cases} \langle id, v_{i+1}\rangle & \text{for } 0 \leq i < n-1, \\ 0 & \text{for } i = n-1, \text{ and} \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \leq i < n.$$

4.3. **Binary operators.** The binary operators require two constructions, one for the case where the left argument has been evaluated and one for the case where the right argument has been evaluated. For each operator op, we define two logspace-computable functions $\mathrm{cir}_\rho(s, op)$ and $\mathrm{cir}_\rho(op, s)$, which compute the circuit given an evaluation $s \in \mathbb{B}^{\|\rho\|}$ of the left and right subformula, respectively.

For the Boolean operators, the two functions are the same, e.g., $\mathrm{cir}_\rho(\vee, s) = \mathrm{cir}_\rho(s, \vee) = \langle G, l, I, O\rangle$, where $n = \|\rho\|$, $I = v_0, \ldots, v_{n-1}$, $O = o_0, \ldots, o_{n-1}$, $G = \{v_0, \ldots, v_{n-1}\} \cup \{o_0, \ldots, o_{n-1}\}$, and

$$l(o_i) = \begin{cases} \langle id, v_i\rangle & \text{for } s_i = 0 \text{ and } 0 \leq i < n, \\ 1 & \text{for } s_i = 1 \text{ and } 0 \leq i < n, \text{ and} \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \leq i < n.$$

For the unbounded temporal operators, the constructions are derived from the expansion laws of the logic, such as $\phi_l \,\mathrm{U}\, \phi_r \equiv \phi_r \vee (\phi_l \wedge \mathrm{X}^\exists (\phi_l \,\mathrm{U}\, \phi_r))$ for the unbounded Until operator. The expanded formula is transformed into a transducer circuit by substituting constants for evaluated subformulas and variable gates for unevaluated subformulas. E.g. $\mathrm{cir}_\rho(\mathrm{U}, s) = \langle G, l, I, O\rangle$, where $n = \|\rho\|$, $I = v_0, \ldots, v_{n-1}$, $O = o_0, \ldots, o_{n-1}$, $G = \{v_0, \ldots, v_{n-1}\} \cup \{o_0, \ldots, o_{n-1}\}$, and

$$l(o_i) = \begin{cases} \langle and, v_i, o_{i+1}\rangle & \text{for } 0 \leq i < n-1 \text{ and } s_i = 0, \\ 1 & \text{for } 0 \leq i < n-1 \text{ and } s_i = 1, \\ s_{n-1} & \text{for } i = n-1, \text{ and} \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \leq i < n,$$

and $\mathrm{cir}_\rho(s, \mathrm{U}) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle or, v_i, o_{i+1}\rangle & \text{for } 0 \leq i < n-1 \text{ and } s_i = 1, \\ \langle id, v_i\rangle & \text{for } 0 \leq i < n-1 \text{ and } s_i = 0, \\ \langle id, v_{n-1}\rangle & \text{for } i = n-1, \text{ and} \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \leq i < n.$$

The most difficult part of the construction is the translation for the bounded operators, which we now present in detail for the bounded Until operator U_b. Figure 6 illustrates the construction of $\mathrm{cir}_\rho(s, \mathrm{U}_b)$ for a valuation $s = 0, 1, 0, 1, 1, 1, 0, 1$ of the left subformula. The gates indexed by i, j compute the value of the formula at position i and "remaining" bound $b - j$. If, at some position, the left subformula evaluates to 0, then the formula simplifies to

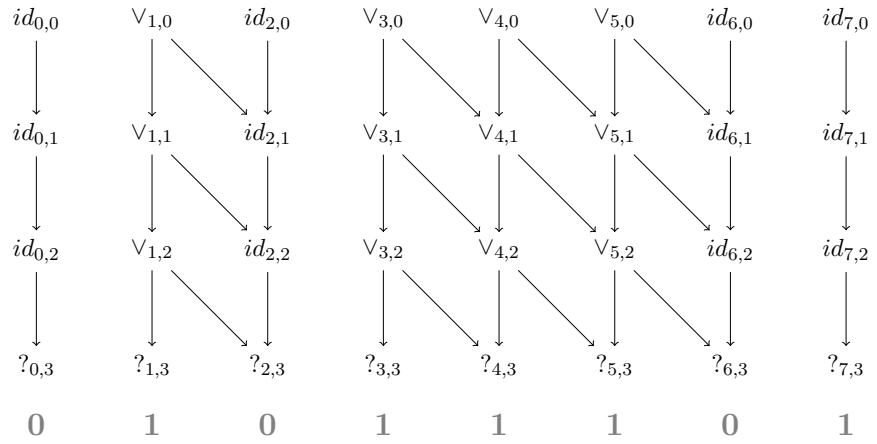

Figure 6: The circuit $\mathrm{cir}_\rho(s, \mathrm{U}_3)$ for $\|\rho\| = 8$. The bottom line shows an example evaluation $s = 0, 1, 0, 1, 1, 1, 0, 1$ of the left subformula.

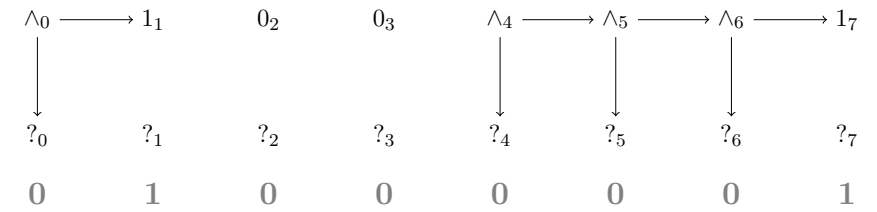

Figure 7: The circuit $\mathrm{cir}_\rho(\mathrm{U}_3, s)$ for $\|\rho\| = 8$. The bottom line shows an example evaluation $s = 0, 1, 0, 0, 0, 0, 0, 1$ of the right subformula.

the right subformula, independently of the remaining bound. This results in vertical edges in the circuit. If the left subformula evaluates to 1, then the formula is true if it is either true for bound $j-1$ in position $i+1$ or for bound $j-1$ in position i. In the circuit, this is computed as a disjunction of the vertical and the diagonal neighbor.

We define $\mathrm{cir}_\rho(s, \mathrm{U}_b) = \langle G, l, I, O\rangle$, where $G = \{v_{i,j} \mid 0 \le i < n, 0 \le j \le b\}$, $I = v_{0,b}, \ldots, v_{n-1,b}$, $O = v_{0,0}, \ldots, v_{n-1,0}$, and

$$l(v_{i,j}) = \begin{cases} \langle id, v_{i,j+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } j < b \text{ and } s_i = 0, \\ \langle or, v_{i,j+1}, v_{i+1,j+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } j < b \text{ and } s_i = 1, \\ \langle id, v_{i,j+1}\rangle & \text{for } i = n-1 \text{ and } j < b, \text{ and} \\ ? & \text{for } j = b. \end{cases}$$

The construction of $\mathrm{cir}_\rho(\mathrm{U}_3, s)$ for the valuation $s = 0, 1, 0, 0, 0, 0, 0, 1$ of the right subformula is illustrated in Figure 7. Here, the gates indexed by i compute the value of the formula at position i. If the right subformula evaluates to 1 in position i, then the value of the formula is 1 in position i, and is computed by the conjunction over the values of the left subformulas in positions $i-b$ to $i-1$. Further to the left from $i-b$, the value is 0 until another 1 occurs in the valuation of the right subformula.

The circuit is therefore defined as $\mathrm{cir}_\rho(\mathrm{U}_b, s) = \langle G, l, I, O\rangle$, where $I = v_0, \ldots, v_{n-1}$, $O = o_0, \ldots, o_{n-1}$, $G = \{v_0, \ldots, v_{n-1}\} \cup \{o_0, \ldots, o_{n-1}\}$, and

$$l(o_i) = \begin{cases} 1 & \text{for } 0 \leq i < n \text{ and } s_i = 1, \\ 0 & \text{for } 0 \leq i < n \text{ and } \forall j, i \leq j < \min(i+b, n).s_j = 0, \\ \langle \mathit{and}, v_i, o_{i+1}\rangle & \text{for } 0 \leq i < n-1,\ s_i = 0,\ \exists j, i < j < \min(i+b, n).s_j = 1, \text{ and} \end{cases}$$

$l(v_i) = ?$ for $0 \leq i < n$.

We conclude the section with a lemma that formally states the existence of the logspace-computable function cir_ρ with the required properties. The complete construction of cir_ρ is provided in the appendix.

Lemma 4.1. *Let ϕ and ψ formulas and p an atomic proposition. Let ρ a path and $s, t \in \mathbb{B}^{\|\rho\|}$ with $s = \phi(\rho)$ and $t = \psi(\rho)$. There is an logspace-computable function cir_ρ mapping its arguments to evaluated one-input-face planar transducer circuits such that $p(\rho) = \mathrm{cir}_\rho(p)()$, $(\mathit{op}\,\phi)(\rho) = f_{\mathrm{cir}_\rho(\mathit{op})}(s)$ for $\mathit{op} \in \{\mathrm{X}^\exists, \mathrm{X}^\forall, \mathrm{Y}^\exists, \mathrm{Y}^\forall\}$, and $(\phi\,\mathit{op}\,\psi)(\rho) = f_{\mathrm{cir}_\rho(s,\mathit{op})}(t)$ and $(\phi\,\mathit{op}\,\psi)(\rho) = f_{\mathrm{cir}_\rho(\mathit{op},t)}(s)$ for $\mathit{op} \in \{\vee, \wedge, \mathrm{U}, \mathrm{U}_b, \mathrm{R}, \mathrm{R}_b, \mathrm{S}, \mathrm{S}_b, \mathrm{T}, \mathrm{T}_b\}$.* □

5. Parallel Tree Contraction for Path Checking

The parallel path checking algorithm for BLTL+Past formulas is based on a bottom-up evaluation of the formula converted to positive normal form starting with the atomic propositions. The central data structure is a binary tree, called the *contraction tree*, that keeps track of the dependencies between the different evaluation steps. Initially, the contraction tree corresponds to the formula tree where each unary node (due to $\mathrm{X}^\exists$, $\mathrm{X}^\forall$, $\mathrm{Y}^\exists$, and $\mathrm{Y}^\forall$ operators) has been merged into a single node with its unique child node. The evaluation of the formula is performed by *contraction steps,* which contract a node that has already been evaluated with its parent into a new edge from its sibling to its parent. The resulting edge is labeled by a planar circuit that represents the partially evaluated subformula.

Since no child needs to wait for the evaluation of its sibling before it can be contracted with its parent, a constant portion of the nodes can be contracted in parallel, and, within logarithmic time, the tree is evaluated to a single constant circuit. We now describe and analyze this process in more detail.

5.1. **Contraction tree.** Given a formula ϕ in positive normal form and a path ρ, let $\phi_0, \ldots, \phi_{m-1}$ be the subformulas of ϕ with $\phi_0 = \phi$. A contraction tree is an edge labeled tree $\mathcal{T} = \langle T, t, l\rangle$ where $T \subseteq \{\phi_0, \ldots, \phi_{m-1}\} \cup \{\mathit{root}\}$, $t \subseteq \{\langle \phi_i, \phi_j\rangle \mid \phi_j \text{ is a subformula of } \phi_i\} \cup \{\langle \mathit{root}, \phi\rangle\}$, and l is a mapping that labels each edge of $\mathcal{T}$ with an evaluated one-input-face planar transducer circuit, such that the following conditions hold:

(1) $\mathcal{T} \setminus \{\mathit{root}\}$ is an ordered, rooted, regular, binary tree,
(2) all edge labels of $\mathcal{T}$ are evaluated one-input-face planar transducer circuits of arity $\|\rho\|$, all leaves are atomic propositions, and
(3) for $\tau = \langle \phi_i, \phi_j\rangle \in t$ it holds that $f_{l(\tau)}(\phi_j(\rho)) = \psi(\rho)$, where ψ is the direct subformula of ϕ_i that has ϕ_j as a subformula. Further, for the unique edge $\tau = \langle \mathit{root}, \phi_j\rangle \in t$ it holds that $f_{l(\tau)}(\phi_j(\rho)) = \phi(\rho)$.

The special node *root* and the corresponding edge $\langle root, \phi\rangle$ were added solely for technical reasons.

The first condition ensures that the overall contraction process performs in a logarithmic number of parallel steps. The second condition provides the preconditions for a single contraction step. Namely, the compositionality of the constructed circuits and the complexity of logDCFL. The third condition states the induction hypothesis for the soundness of the whole algorithm: When a transducer circuit is attached to an edge of the contraction tree, it encodes the semantics of all partially evaluated subformulas contracted into that edge.

5.2. **Initialization step.** The initial contraction tree $\mathcal{T} \setminus \{root\}$ is the formula tree ϕ where each unary node (due to $\mathrm{X}^\exists$, $\mathrm{X}^\forall$, $\mathrm{Y}^\exists$, and $\mathrm{Y}^\forall$ operators) has been merged with its unique child node. The corresponding new parent edge is labeled by the transducer circuit that results from composing the circuits produced by applying cir_ρ to the corresponding BLTL+Past operators of the eliminated nodes.

Lemma 5.1. *Given a formula ϕ in positive normal form and a path ρ, a contraction tree $\mathcal{T}$ can be constructed from ϕ and ρ by an logspace restricted Turing machine.*

Proof. Define $\mathrm{parent}(\chi)$ to be the subformula ψ of ϕ such that χ is the maximal subformula of ψ in ϕ. Let $\mathcal{T} = \langle T, t, l\rangle$ with $T = \{\phi_i \mid \phi_i \text{ is not of the form } \mathrm{X}^\exists\psi, \mathrm{X}^\forall\psi, \mathrm{Y}^\exists\psi, \text{ or } \mathrm{Y}^\forall\psi, 0 \le i < m\} \cup \{root\}$, $t = \{\langle \phi_i, \phi_j\rangle \in T \times T \mid \phi_j \text{ is a maximal subformula in } T \text{ of } \phi_i\} \cup \{\langle root, \phi\rangle\}$, and for $\tau \in t$,

$$l(\tau) = \begin{cases} id & \text{for } \tau = \langle root, \phi\rangle, \\ c(\tau) & \text{otherwise,} \end{cases}$$

where for $0 \le i < m$,

$$c(\langle \phi_i, x\rangle) = \begin{cases} \mathrm{cir}_\rho(\mathrm{X}^\exists) \circ c(\mathrm{parent}(\phi_i)) & \text{for } \mathrm{parent}(\phi_i) = \mathrm{X}^\exists\phi_i, \\ \mathrm{cir}_\rho(\mathrm{X}^\forall) \circ c(\mathrm{parent}(\phi_i)) & \text{for } \mathrm{parent}(\phi_i) = \mathrm{X}^\forall\phi_i, \\ \mathrm{cir}_\rho(\mathrm{Y}^\exists) \circ c(\mathrm{parent}(\phi_i)) & \text{for } \mathrm{parent}(\phi_i) = \mathrm{Y}^\exists\phi_i, \\ \mathrm{cir}_\rho(\mathrm{Y}^\forall) \circ c(\mathrm{parent}(\phi_i)) & \text{for } \mathrm{parent}(\phi_i) = \mathrm{Y}^\forall\phi_i, \\ id & \text{otherwise,} \end{cases}$$

where id is the identity transducer circuit of arity $\|\rho\|$.

In $\mathcal{T}$, all simple paths (due to $\mathrm{X}^\exists$, $\mathrm{X}^\forall$, $\mathrm{Y}^\exists$, and $\mathrm{Y}^\forall$ operators) have been collapsed into single edges. This ensures that $\mathcal{T} \setminus \{root\}$ is an ordered, rooted, regular, binary tree. The circuits $\mathrm{cir}_\rho(\mathrm{X}^\exists)$, $\mathrm{cir}_\rho(\mathrm{X}^\forall)$, $\mathrm{cir}_\rho(\mathrm{Y}^\exists)$, and $\mathrm{cir}_\rho(\mathrm{Y}^\forall)$ are evaluated one-input-face planar transducer circuits that do not contain any constants. Hence, any number of these circuits can be composed resulting in an evaluated one-input-face planar transducer circuit. The composition of planar transducer circuits can be performed in logarithmic space. The mapping c is defined recursively above. However, it is easy to see that the whole procedure can be performed iteratively in logarithmic space in the size of ρ plus the size of ϕ. From the above, the first and the second condition for a contraction tree are clear. The third condition is obtained by applying Lemma 4.1 to the construction. □

5.3. **Contraction step.** In the following, we describe the contraction of the tree $\mathcal{T}$. During a contraction step, a node that is labeled by a constant circuit is merged with its parent node. The resulting node is contracted into the edge from its sibling to its grandparent.

Lemma 5.2. *Let ϕ_i a node of a contraction tree $\mathcal{T}$ with child nodes ϕ_j and ϕ_k and parent node p. Assume ϕ_j to be a leaf. Let s be the evaluation of $cir_\rho(\phi_j) \circ l(\langle \phi_i, \phi_j \rangle)$. Let $\mathcal{T}' = \langle T', t', l' \rangle$, where*

$$T' = T \setminus \{\phi_j, \phi_i\},$$
$$t' = t \cup \{\langle \phi_p, \phi_k \rangle\} \setminus \{\langle \phi_i, \phi_j \rangle, \langle \phi_i, \phi_k \rangle, \langle \phi_p, \phi_i \rangle\},$$
$$l'(\phi_k) = \begin{cases} \textit{evaluation of } l(\langle \phi_i, \phi_k \rangle) \circ \mathrm{cir}_\rho(f_s(), \phi_i) \circ l((\phi_i, p)) & \textit{if } \phi_j \textit{ is the left child of } \phi_i, \\ \textit{evaluation of } l(\langle \phi_i, \phi_k \rangle) \circ \mathrm{cir}_\rho(\phi_i, f_s()) \circ l((\phi_i, p)) & \textit{if } \phi_j \textit{ is the right child of } \phi_i, \end{cases}$$
$$l'(x) = l(x) \textit{ for } x \neq \phi_k.$$

$\mathcal{T}'$ is a contraction tree and can be computed in logDCFL.

Proof. First, note that by construction of $\mathcal{T}$ it holds that $\phi_i, \phi_j, \phi_k \neq root$. Clearly, if $\mathcal{T} \setminus \{root\}$ is an ordered, rooted, regular, binary tree then $\mathcal{T}' \setminus \{root\}$ is an ordered, rooteted, regular, binary tree, as well. By construction of $\mathcal{T}'$ a leaf in $\mathcal{T}'$ is a leaf in $\mathcal{T}$ as well. Thus, because $\mathcal{T}$ is a contraction tree, each leaf in $\mathcal{T}'$ is an atomic proposition. Since ϕ_j is a leaf ϕ_j is an atomic proposition. Due to Lemma 4.1 $\mathrm{cir}_\rho(\phi_j)$ can be composed with $l(\langle \phi_i, \phi_j \rangle)$ resulting in an one-input-face planar circuit that can be evaluated in logDCFL. Thus s is a constant circuit of arity $\|\rho\|$ and $\mathrm{cir}_\rho(f_s(), \phi_i)$ (respectively $\mathrm{cir}_\rho(\phi_i, f_s())$) is well defined and of arity $\|\rho\|$. Because $\mathcal{T}$ is a contraction tree and by Lemma 3.3, $l'(\langle p, \phi_k \rangle)$ is an evaluated one-input-face planar transducer circuit. By the definition of $\circ$ the input arity of $l'(\langle p, \phi_k \rangle)$ is the input arity of $l(\langle \phi_i, \phi_k \rangle)$ and the output arity of $l'(\langle p, \phi_k \rangle)$ is the output arity of $l(\langle p, \phi_i \rangle)$. Because $\mathcal{T}$ is a contraction tree this arity is $\|\rho\|$ in both cases. All remaining edge labels of $\mathcal{T}'$ inherit the arities from $\mathcal{T}$. Considering the edge $\langle p, \phi_k \rangle \in t'$, the third condition for a contraction tree holds, since $\mathcal{T}$ is a contraction tree, and due to the definition of $\circ$, and because of Lemma 4.1. For all other edges, the property is directly inherited from $\mathcal{T}$. The computation of $\mathcal{T}'$ is in logDCFL because of Lemma 3.3 and Lemma 4.1. □

5.4. **The path checking algorithm.** Applying Lemma 5.1 and Lemma 5.2 to ϕ and ρ, we can use Algorithm 2.3 to obtain an AC^1(logDCFL) solution to the path checking problem. This proves our main theorem:

Theorem 3. Given a BLTL+Past formula ϕ and a path ρ, convert ϕ into positive normal form using only logarithmic space. A contraction tree $\mathcal{T}$ is initialized from ϕ in logarithmic space by use of Lemma 5.1 and then Algorithm 2.3 is applied to $\mathcal{T}$ with the contraction step defined in Section 5.3. Note that the extra *root* node and the edge $\langle root, \phi \rangle$ in $\mathcal{T}$ do not influence the performance of Algorithm 2.3. The algorithm terminates when there is only a single leaf node n and a single edge $\langle root, n \rangle$ left in the contraction tree. By Lemma 5.2, the contraction algorithm performes in AC^1(logDCFL). The value of the first output gate of the evaluation of the circuit $c = \mathrm{cir}_\rho(n) \circ l(\langle root, n \rangle)$ is the result. By Lemma 5.2, c is evaluated and one-input-face and can hence be evaluated in logDCFL. The whole construction can be executed within AC^1(logDCFL) in $\|\phi\| + \|\rho\|$. Using Lemma 2.2, we can assume that any bound occurring in ϕ has at most size $\|\rho\|$. The sum of the bounds is thus polynomial

in the size of ϕ (without bounds) and the length of ρ. Thus, the overall complexity of $\mathsf{AC}^1(\mathsf{logDCFL})$ is independent of the encoding of the bounds in ϕ. □

Since LTL and $LTL+Past$ both are subsets of $BLTL+Past$ Theorem 1 and Theorem 2 are obtained as corollaries of Theorem 3.

6. Conclusions

We have presented a positive answer to the question whether LTL can be checked efficiently in parallel on finite paths by giving an $\mathsf{AC}^1(\mathsf{logDCFL})$ algorithm for checking formulas of the extended logic BLTL+Past over finite paths. This result is 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 [26]. While other extensions of LTL, for example with Chop or Past+Now, immediately render the path checking problem P-complete and, hence, inherently sequential [26], 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, tools that translate PSL assertions to hardware-based monitors [7, 5, 11] can immediately apply our construction to evaluate subformulas consisting of bounded and past operators in parallel rather than sequentially. Similarly, monitoring tools based on LTL+Past can buffer constant chunks of the input and then evaluate the buffered input in parallel using our construction.

The capability of our algorithm to absorb the exponential succinctness of past and bounds is due to 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 expect that the use of planar circuits as a data structure in parallel verification algorithms, following the pattern of our construction, will find applications in other model checking problems as well.

There are several open questions that deserve further attention. There is still a gap between $\mathsf{AC}^1(\mathsf{logDCFL})$ and the known lower bound, NC^1. There is some hope to further reduce the upper bound towards NC^1, the currently known lower bound, because our construction relies on the algorithm by Chakraborty and Datta (cf. Theorem 3.1) for evaluating monotone Boolean planar 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 $\mathsf{logDCFL}$. An intriguing question along the way is whether the path checking complexities of LTL and BLTL+Past are actually the same: while they are both in NC, the circuits resulting from BLTL+Past formulas seem to be combinatorially more complex. Finally, an interesting challenge is to exploit the apparent "cheapness" of the BLTL+Past path checking problem beyond parallelization, for example in memory-efficient algorithms.

References

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

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

[3] Cyrille Artho, Howard Barringer, Allen Goldberg, Klaus Havelund, Sarfraz Khurshid, Mike Lowry, Corina Pasareanu, Grigore Rosu, Koushik Sen, Willem Visser, and Rich Washington. Combining test case generation and runtime verification. *Theoretical Computer Science*, 336(2-3):209 – 234, 2005.
[4] D.A.M. Barrington, Chi-Jen Lu, P.B. Miltersen, and S. Skyum. On monotone planar circuits. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity (COCO '99)*, pages 24–31, Washington, DC, USA, 1999. IEEE Computer Society.
[5] 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.
[6] T. Chakraborty and S. Datta. One-input-face MPCVP is hard for L, but in LogDCFL. In *Proc. FSTTCS*, volume 4337 of *LNCS*, pages 57–68. Springer, 2006.
[7] 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 *Proc. ISQED'05*, pages 310–315. IEEE Computer Society, 2005.
[8] A.L. Delcher and S.R. Kosaraju. An NC algorithm for evaluating monotone planar circuits. *SIAM J. Comput.*, 24(2):369–375, 1995.
[9] S. Demri and Ph. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Inf. Comput.*, 174(1):84–103, 2002.
[10] P.W. Dymond and S.A. Cook. Complexity theory of parallel time and hardware. *Information and Computation*, 80(3):205–226, 1989.
[11] B. Finkbeiner and L. Kuhtz. Monitor circuits for LTL with bounded and unbounded future. In *Proc. RV'09*, LNCS. Springer, 2009.
[12] B. Finkbeiner and H.B. Sipma. Checking finite traces using alternating automata. *Formal Methods in System Design*, 24:101–127, 2004.
[13] L.M. Goldschlager. A space efficient algorithm for the monotone planar circuit value problem. *Inf. Process. Lett.*, 10(1):25–27, 1980.
[14] K. Havelund and G. Roşu. Efficient monitoring of safety properties. *STTT*, 6(2):158–173, 2004.
[15] IEEE Std 1850-2007. *Property Specification Language (PSL)*. IEEE, New York, 2007.
[16] 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.
[17] D.S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 67–161. MIT Press, 1990.
[18] Richard M. Karp and Vijaya 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.
[19] S. Rao Kosaraju and Arthur 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.
[20] S.R. Kosaraju. On parallel evaluation of classes of circuits. In *Proc. FSTTCS*, volume 472 of *LNCS*, pages 232–237. Springer, 1990.
[21] L. Kuhtz. *Model Checking Finite Paths and Trees*. PhD thesis, Universität des Saarlandes, 2010.
[22] L. Kuhtz and B. Finkbeiner. LTL path checking is efficiently parallelizable. In *Proc. ICALP'09*, volume 5556 of *LNCS*, pages 235 – 246. Springer, 2009.
[23] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proc. LICS*, pages 383–392. IEEE Computer Society, 2002.
[24] O. Lichtenstein, A. Pnueli, and L.D. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218, London, UK, 1985. Springer.
[25] N. Limaye, M. Mahajan, and J.M.N. Sarma. Evaluating monotone circuits on cylinders, planes and tori. In B. Durand and W. Thomas, editors, *Proc. STACS*, volume 3884 of *LNCS*, pages 660–671. Springer, 2006.
[26] N. Markey and Ph. Schnoebelen. Model checking a path (preliminary report). In *Proc. CONCUR*, volume 2761 of *LNCS*, pages 251–265. Springer, 2003.
[27] H. Petersen. Decision problems for generalized regular expressions. In *2nd International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures, London(Ontario)*, pages 22–29, 2000.

[28] H. Petersen. The membership problem for regular expressions with intersection is complete in LOGCFL. In *Proc. STACS*, volume 2285 of *LNCS*, pages 513–522. Springer, 2002.
[29] Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer, 1999.
[30] H. Yang. An NC algorithm for the general planar monotone circuit value problem. In *Proc. SPDP*, pages 196–203. IEEE Computer Society, 1991.
[31] H.L.S Younes and R.G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. CAV*, volume 2404 of *LNCS*. Springer, 2002.

Appendix: Construction of cir_ρ

Let $n = \|\rho\|$. Let $b \in \mathbb{N}$. Let $I = v_0, \ldots, v_{n-1}$ and $O = o_0, \ldots, o_{n-1}$. Let $G = I \cup O$. Let $H = \{g_{i,j} \mid 0 \le i < n, 0 \le j \le b\}$. Let $O_H = g_{0,0}, \ldots, g_{n-1,0}$. Let $I_H = g_{0,b}, \ldots, g_{n-1,b}$.

For an atomic proposition p $\mathrm{cir}_\rho(p) = \langle O, l, \varepsilon, O\rangle$, where $l(o_i) = 1$ iff $p \in \rho_i$ and ε denotes an empty input sequence.

$\mathrm{cir}_\rho(\vee, s) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle id, v_i\rangle & \text{for } s_i = 0 \text{ and } 0 \le i < n, \\ 1 & \text{for } s_i = 1 \text{ and } 0 \le i < n, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(s, \vee) = \mathrm{cir}_\rho(\vee, s)$;

$\mathrm{cir}_\rho(\wedge, s) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle id, v_i\rangle & \text{for } s_i = 1 \text{ and } 0 \le i < n, \\ 0 & \text{for } s_i = 0 \text{ and } 0 \le i < n, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(s, \wedge) = \mathrm{cir}_\rho(\wedge, s)$;

$\mathrm{cir}_\rho(\mathrm{X}^\exists) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle id, v_{i+1}\rangle & \text{for } 0 \le i < n-1, \\ 0 & \text{for } i = n-1, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(\mathrm{X}^\forall) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle id, v_{i+1}\rangle & \text{for } 0 \le i < n-1, \\ 1 & \text{for } i = n-1, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(\mathrm{U}, s) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle and, v_i, o_{i+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } s_i = 0, \\ 1 & \text{for } 0 \le i < n-1 \text{ and } s_i = 1, \\ s_{n-1} & \text{for } i = n-1, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(s, \mathrm{U}) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle or, v_i, o_{i+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } s_i = 1, \\ \langle id, v_i\rangle & \text{for } 0 \le i < n-1 \text{ and } s_i = 0, \\ \langle id, v_{n-1}\rangle & \text{for } i = n-1, \end{cases}$$

$l(v_i) = ?$ for $0 \le i < n$;

$\mathrm{cir}_\rho(\mathrm{R}, s) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle or, v_i, o_{i+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } s_i = 1, \\ 0 & \text{for } 0 \le i < n-1 \text{ and } s_i = 0, \\ s_{n-1} & \text{for } i = n-1, \end{cases}$$

$l(v_i) = ?$ for $0 \le i < n$;

$\mathrm{cir}_\rho(s, \mathrm{R}) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle and, v_i, o_{i+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } s_i = 0, \\ \langle id, v_i\rangle & \text{for } 0 \le i < n-1 \text{ and } s_i = 1, \\ \langle id, v_{n-1}\rangle & \text{for } i = n-1, \end{cases}$$

$l(g) = ?$ for $g \in I$;

$\mathrm{cir}_\rho(\mathrm{U}_b, s) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} 1 & \text{for } 0 \le i < n \text{ and } s_i = 1, \\ 0 & \text{for } 0 \le i < n \text{ and } \forall j, i \le j < \min(i+b, n).s_j = 0, \\ \langle and, v_i, o_{i+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } s_i = 0 \text{ and } \exists j, i < j < \min(i+b, n).s_j = 1, \end{cases}$$

$l(v_i) = ?$ for $0 \le i < n$;

$\mathrm{cir}_\rho(s, \mathrm{U}_b) = \langle H, l, I_H, O_H\rangle$, where

$$l(g_{i,j}) = \begin{cases} \langle id, g_{i,j+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } j < b \text{ and } s_i = 0, \\ \langle or, g_{i,j+1}, g_{i+1,j+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } j < b \text{ and } s_i = 1, \\ \langle id, g_{i,j+1}\rangle & \text{for } i = n-1 \text{ and } j < b, \\ ? & \text{for } j = b, \end{cases}$$

$\mathrm{cir}_\rho(\mathrm{R}_b, s) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} 0 & \text{for } 0 \le i < n \text{ and } s_i = 0, \\ 1 & \text{for } 0 \le i < n \text{ and } \forall j, i \le j < \min(i+b, n).s_j = 1, \\ \langle or, v_i, o_{i+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } s_i = 1 \text{ and } \exists j, i < j < \min(i+b, n).s_j = 0, \end{cases}$$

$l(v_i) = ?$ for $0 \le i < n$;

$\mathrm{cir}_\rho(s, \mathrm{R}_b) = \langle H, l, I_H, O_H\rangle$, where

$$l(g_{i,j}) = \begin{cases} \langle id, g_{i,j+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } j < b \text{ and } s_i = 1, \\ \langle and, g_{i,j+1}, g_{i+1,j+1}\rangle & \text{for } 0 \le i < n-1 \text{ and } j < b \text{ and } s_i = 0, \\ \langle id, g_{i,j+1}\rangle & \text{for } i = n-1 \text{ and } j < b, \\ ? & \text{for } j = b; \end{cases}$$

$\mathrm{cir}_\rho(\mathrm{Y}^\exists) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle id, v_{i-1}\rangle & \text{for } 0 < i < n, \\ 0 & \text{for } i = 0, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(\mathrm{Y}^\forall) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle id, v_{i-1}\rangle & \text{for } 0 < i < n, \\ 1 & \text{for } i = 0, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(\mathrm{S}, s) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle and, v_i, o_{i-1}\rangle & \text{for } 0 < i < n \text{ and } s_i = 0, \\ 1 & \text{for } 0 < i < n \text{ and } s_i = 1, \\ s_0 & \text{for } i = 0, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(s, \mathrm{S}) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle or, v_i, o_{i-1}\rangle & \text{for } 0 < i < n \text{ and } s_i = 1, \\ \langle id, v_i\rangle & \text{for } 0 < i < n \text{ and } s_i = 0, \\ \langle id, v_0\rangle & \text{for } i = 0, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(\mathrm{T}, s) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle or, v_i, o_{i-1}\rangle & \text{for } 0 < i < n \text{ and } s_i = 1, \\ 0 & \text{for } 0 < i < n \text{ and } s_i = 0, \\ s_0 & \text{for } i = 0, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(s, \mathrm{T}) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} \langle and, v_i, o_{i-1}\rangle & \text{for } 0 < i < n \text{ and } s_i = 0, \\ \langle id, v_i\rangle & \text{for } 0 < i < n \text{ and } s_i = 1, \\ \langle id, v_0\rangle & \text{for } i = 0, \end{cases}$$
$$l(g) = ? \text{ for } g \in I;$$

$\mathrm{cir}_\rho(\mathrm{S}_b, s) = \langle G, l, I, O\rangle$, where

$$l(o_i) = \begin{cases} 1 & \text{for } 0 \le i < n \text{ and } s_i = 1, \\ 0 & \text{for } 0 \le i < n \text{ and } \forall j, i \ge j > \max(i-b, -1).s_j = 0, \\ \langle and, v_i, o_{i-1}\rangle & \text{for } 0 < i < n \text{ and } s_i = 0 \text{ and } \exists j, i > j > \max(i-b, -1).s_j = 1, \end{cases}$$
$$l(v_i) = ? \text{ for } 0 \le i < n;$$

$\mathrm{cir}_\rho(s, \mathrm{S}_b) = \langle H, l, I_H, O_H \rangle$, where

$$l(g_{i,j}) = \begin{cases} \langle id, g_{i,j+1} \rangle & \text{for } 0 < i < n \text{ and } j < b \text{ and } s_i = 0, \\ \langle or, g_{i,j+1}, g_{i-1,j+1} \rangle & \text{for } 0 < i < n \text{ and } j < b \text{ and } s_i = 1, \\ \langle id, g_{0,j+1} \rangle & \text{for } j < b, \\ ? & \text{for } j = b; \end{cases}$$

$\mathrm{cir}_\rho(\mathrm{T}_b, s) = \langle G, l, I, O \rangle$, where

$$l(o_i) = \begin{cases} 0 & \text{for } 0 \leq i < n \text{ and } s_i = 0, \\ 1 & \text{for } 0 \leq i < n \text{ and } \forall j, i \geq j > \min(i - b, -1).s_j = 1, \\ \langle or, v_i, o_{i-1} \rangle & \text{for } 0 < i < n \text{ and } s_i = 1 \text{ and } \exists j, i > j > \max(i - b, -1).s_j = 0, \end{cases}$$

$l(v_i) = ?$ for $0 \leq i < n$;

$\mathrm{cir}_\rho(s, \mathrm{T}_b) = \langle H, l, I_H, O_H \rangle$, where

$$l(g_{i,j}) = \begin{cases} \langle id, g_{i,j+1} \rangle & \text{for } 0 < i < n \text{ and } j < b \text{ and } s_i = 1, \\ \langle and, g_{i,j+1}, g_{i-1,j+1} \rangle & \text{for } 0 < i < n \text{ and } j < b \text{ and } s_i = 0, \\ \langle id, g_{0,j+1} \rangle & \text{for } j < b, \\ ? & \text{for } j = b. \end{cases}$$