LTL Path Checking is Efficiently Parallelizable*

Lars Kuhtz and Bernd Finkbeiner

Universität des Saarlandes
66123 Saarbrücken, Germany
{kuhtz|finkbeiner}@cs.uni-sb.de

Abstract. We present an $\mathsf{AC}^1(\mathsf{logDCFL})$ algorithm for checking LTL formulas over finite paths, thus establishing that the problem can be efficiently parallelized. Our construction provides a foundation for the parallelization of various applications in monitoring, testing, and verification.

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 [12,10,6,1,4], 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 [2] and in several static verification techniques, notably in Monte-Carlo-based probabilistic verification, where large numbers of randomly generated sample paths are analyzed [22].

Somewhat surprisingly, given the widespread use of LTL, the complexity of the path checking problem is still open [18]. The established upper bound is P: The algorithms in the literature traverse the path sequentially (cf. [10,18,12]); 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 [8], the complexity of evaluating 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 [13], an extension of LTL, has become part of the hardware description language VHDL, and several tools [6,4] 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 this paper, 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 [11,9,15,3,17,5]. Rather than sequentially traversing the path, we consider the circuit that results from unrolling the formula over the path. Figure 1 shows such a circuit for the formula $((a \,\mathsf{U}\, b) \,\mathsf{U}\, (c \,\mathsf{U}\, d)) \,\mathsf{U}\, e$ and a path of length 5.

* 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).

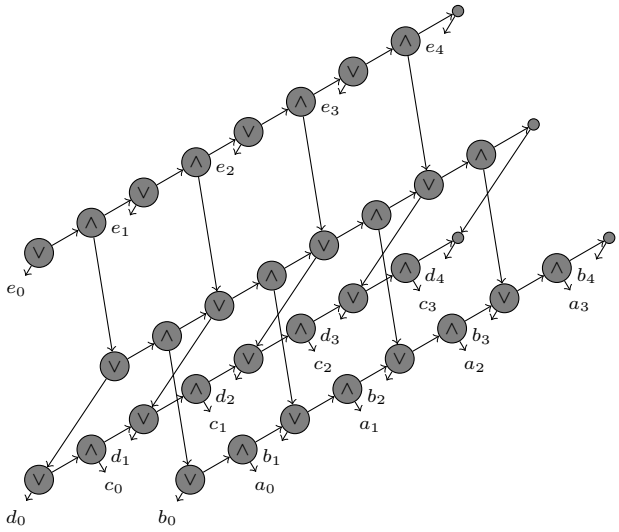

Fig. 1. Circuit resulting from unrolling the LTL formula $((a \,\mathrm{U}\, b) \,\mathrm{U}\, (c \,\mathrm{U}\, d)) \,\mathrm{U}\, e$ over a path ρ of length 5. We denote the value of an atomic proposition p at a path position $i = 0, \ldots, 4$ by p_i. The graph of the circuit has no planar embedding.

Yang [21] and, independently, Delcher and Kosaraju [7] 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. 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 introduce a contraction technique that combines a parent node and its children into a single planar circuit. Simple paths in the tree immediately collapse into a planar circuit; the remaining binary tree is contracted incrementally, until only a single planar circuit remains. The key insight of our solution is that the contraction can be carried out *as soon as one of the children has been evaluated.* Because no evaluated child 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. 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 $\mathsf{AC}^1(\mathsf{logDCFL})$.

1 Preliminaries

Linear-Time Temporal Logic. We consider specifications in linear-time temporal logic (LTL). We apply the usual finite-path semantics with a weak and a strong version of the *Next*-Operator [16]. Let P be a set of atomic propositions. The *formulas* of LTL are defined inductively as follows: For each atomic proposition $p \in P$ p and $\neg p$ are LTL formulas. If ϕ and ψ are LTL formulas, then so are

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

LTL formulas are evaluated over computation paths. A *path* $\rho = \rho_0, \ldots, \rho_{n-1}$ is a 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 $\|\rho\|$. The *suffix* of ρ at position i, $0 \leq i < n$, is denoted by ρ^i. The *empty path* is denoted by ϵ.

Given an LTL formula ϕ, a nonempty path $\rho \neq \epsilon$ satisfies ϕ, denoted by $\rho \models \phi$, if one of the following holds:

- $\phi \in P$ and $\phi \in \rho_0$,
- $\phi = \neg p$ and $p \notin \rho_0$,
- $\phi = \phi_l \wedge \phi_r$ and $\rho \models \phi_l$ and $\rho \models \phi_r$,
- $\phi = \phi_l \vee \phi_r$ and $\rho \models \phi_l$ or $\rho \models \phi_r$,
- $\phi = \mathrm{X}_\exists \psi$ and $\rho^1 \models \psi$ and $\rho^1 \neq \epsilon$,
- $\phi = \mathrm{X}_\forall \psi$ and $\rho^1 \models \psi$ or $\rho^1 = \epsilon$,
- $\phi = \phi_l\, \mathrm{U}\, \phi_r$ and $\exists 0 \leq i < \|\rho\|$ s.t. $\rho^i \models \phi_r$ and $\forall 0 \leq j < i$, $\rho^j \models \phi_l$, or
- $\phi = \phi_l\, \mathrm{R}\, \phi_r$ and $\forall 0 \leq i < \|\rho\|, \rho^i \models \phi_r$ or $\exists 0 \leq j < i$ s.t. $\rho^j \models \phi_l$.

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:

$$\phi_l\, \mathrm{U}\, \phi_r \ \equiv\ \phi_r \vee (\phi_l \wedge \mathrm{X}_\exists\, (\phi_l\, \mathrm{U}\, \phi_r)); \quad \phi_l\, \mathrm{R}\, \phi_r \ \equiv\ \phi_r \wedge (\phi_l \vee \mathrm{X}_\forall\, (\phi_l\, \mathrm{R}\, \phi_r))\ .$$

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

Definition 1 (Path Checking Problem). *The path checking problem for LTL is to decide, for an LTL formula ϕ and a nonempty path ρ, whether $\rho \models \phi$.*

Complexity classes within P. We assume familiarity with the standard complexity classes within P. 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 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 depth $\log^i$, where the depth of a circuit is the length of a longest directed path

in the circuit. AC is defined as $\bigcup_{i\in\mathbb{N}} \mathsf{AC}^i$. Throughout the paper, all circuits are assumed to be uniform in the sense that the circuit for inputs of length n can be generated by a deterministic Turing machine using space $\log(n)$. It holds that

$$\mathsf{L} \subseteq \mathsf{logDCFL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{AC}^2 \subseteq \cdots \subseteq \mathsf{AC} \subseteq \mathsf{P} \ .$$

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

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

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 or with a tuple $\langle \mathit{and}, \mathit{left}, \mathit{right} \rangle$, $\langle \mathit{or}, \mathit{left}, \mathit{right} \rangle$, $\langle \mathit{id}, \mathit{suc} \rangle$, where *left*, *right*, and *suc* are gates.

A gate that is labeled with a Boolean value is called a *constant gate*. For a non-constant gate a labeled with $\langle \mathit{id}, b \rangle$, we say that a *directly depends* on b, denoted by $a \succ b$. Likewise, for a gate a labeled with $\langle \mathit{and}, b, c \rangle$ or $\langle \mathit{or}, b, c \rangle$, a directly depends on b and 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 set of gates G, $\mathrm{const}(G)$ denotes the set of all constant gates in G. If $G = \mathrm{const}(G)$, we call G *constant*.

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. We will often analyze subcircuits which are only well-defined in the context of the full circuit. We call such subcircuits partial circuits: Given a circuit $C = \langle \Gamma, \gamma \rangle$, a *partial circuit* is a circuit $D = \langle \Delta, \delta \rangle$ with $\Delta \subseteq \Gamma$ and $\delta = \gamma|_\Delta$. The gates in $\{g \in \Gamma \setminus \Delta \mid \exists h \in \Delta.\, h \succ g\}$ are called the *variable gates* of D. For a variable gate g of D, we define $\delta(g) = \bot$. If C is clear from the context, we refer to D as Δ.

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 $\gamma(g) = \langle \mathit{and}, l, r \rangle$ and $\gamma'(l) = 0$ or $\gamma'(r) = 0$,
- $\gamma'(g) = 1$ iff $\gamma(g) = \langle \mathit{and}, l, r \rangle$ and $\gamma'(l) = 1$ and $\gamma'(r) = 1$,
- $\gamma'(g) = \langle \mathit{id}, l \rangle$ iff $\gamma(g) = \langle \mathit{and}, l, r \rangle$ and $\gamma'(l) \notin \{0, 1\}$ and $\gamma'(r) = 1$,
- $\gamma'(g) = \langle \mathit{id}, r \rangle$ iff $\gamma(g) = \langle \mathit{and}, l, r \rangle$ and $\gamma'(r) \notin \{0, 1\}$ and $\gamma'(l) = 1$,
- $\gamma'(g) = 0$ iff $\gamma(g) = \langle \mathit{or}, l, r \rangle$ and $\gamma'(l) = 0$ and $\gamma'(r) = 0$,
- $\gamma'(g) = 1$ iff $\gamma(g) = \langle \mathit{or}, l, r \rangle$ and $\gamma'(l) = 1$ or $\gamma'(r) = 1$,
- $\gamma'(g) = \langle \mathit{id}, l \rangle$ iff $\gamma(g) = \langle \mathit{or}, l, r \rangle$ and $\gamma'(l) \notin \{0, 1\}$ and $\gamma'(r) = 0$,
- $\gamma'(g) = \langle \mathit{id}, r \rangle$ iff $\gamma(g) = \langle \mathit{or}, l, r \rangle$ and $\gamma'(r) \notin \{0, 1\}$ and $\gamma'(l) = 0$,
- $\gamma'(g) = \gamma'(s)$ iff $\gamma(g) = \langle \mathit{id}, s \rangle$ 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 full circuit evaluates to a constant circuit; for a partial circuit, a subset of the gates is relabeled: some *and-*/*or-*/*id*-gates are labeled as constant or *id*-gates. In the construction presented in this paper, we evaluate circuits in several stages by evaluating partial circuits. In this process, the evaluation of a partial circuit includes substituting the partial circuit by its evaluation within the full circuit. Since the evaluation of a partial circuit is a local operation, disjoint partial circuits can be evaluated in parallel.

The problem of evaluating monotone planar circuits has been studied extensively in the literature. 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 $\langle V, E \rangle$, where $V = \Gamma \cup X$ and $E = \{\langle a, b \rangle \in V \times V \mid a \succ b\}$. A circuit C is *planar* if there exists an planar embedding of the graph of C. The *input gates* of C are all constant and all variable gates of C. A planar partial circuit is *one-input-face* if there is a planar embedding such that all input gates are located on the outer face.

In the following, we abbreviate *evaluated circuit* as EV and *one-input-face planar* as OIF, using the terms EV and OIF for the circuits as well as for the corresponding property of a circuit. Note that an EV circuit with all variables on the outer face is OIF. The evaluation of full OIF circuits can be parallelized efficiently. We make use of a result by Chakraborty and Datta [5]:

Theorem 1 (Chakraborty and Datta 2006). *The problem of evaluating a full* OIF *circuit is in* logDCFL.

Using standard techniques for partial circuits [15], the theorem generalizes from full to partial circuits:

Corollary 1. *The problem of evaluating an* OIF *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. □

2 From LTL to Circuits

In this section, we provide an L many-one reduction from the path checking problem of LTL to the problem of evaluating monotone Boolean circuits.

Given an LTL formula ϕ and a path ρ, we define a circuit $C(\phi, \rho) = \langle \Gamma, \gamma \rangle$ such that $\rho \models \phi$ if and only if a distinguished result gate $c_{0,0}$ is mapped to `1` in the evaluation of $C(\phi, \rho)$. The circuit is constructed by unrolling ϕ on ρ into a DAG according to the expansion laws.

Definition 2. *Given an LTL formula ϕ and a path ρ, the circuit $C(\phi,\rho) = \langle \Gamma, \gamma\rangle$ is defined as follows. Let $\phi_0,\dots,\phi_{m-1}$ (with $\phi_0 = \phi$) be the subformulas of ϕ and let $\rho = \rho_0,\dots,\rho_{n-1}$. The set of gates $\Gamma = \bigcup_{\substack{i=0,\dots,m-1\\ j=0,\dots,n-1}} C_{i,j}$ contains for each subformula ϕ_i and each path position $0 \le j < n$ the set $C_{i,j}$ of gates defined below:*

- $C_{i,j} = \{c_{i,j}\}$ *if $j = n-1$ or if ϕ_i is either an atomic proposition, a negated atomic proposition, a conjunction, a disjunction, an $\mathrm{X}_\exists$-formula, or an $\mathrm{X}_\forall$-formula, and*
- $C_{i,j} = \{c_{i,j}, c'_{i,j}\}$ *if $0 \le j < n-1$ and ϕ_i is either an* U*-formula or an* R*-formula,*

where $c_{i,j}, c'_{i,j}, i = 0,\dots,m-1, j = 0,\dots,n-1$ are distinct gates. The gates are labeled as follows. For $0 \le j < n-1$:

- $\gamma(c_{i,j}) = \langle or, c_{r,j}, c'_{i,j}\rangle$ *and*
  $\gamma(c'_{i,j}) = \langle and, c_{l,j}, c_{i,j+1}\rangle$ *for $\phi_i = \phi_l \,\mathrm{U}\, \phi_r$,*
- $\gamma(c_{i,j}) = \langle and, c_{r,j}, c'_{i,j}\rangle$ *and*
  $\gamma(c'_{i,j}) = \langle or, c_{l,j}, c_{i,j+1}\rangle$ *for $\phi_i = \phi_l \,\mathrm{R}\, \phi_r$, and*
- $\gamma(c_{i,j}) = \langle id, c_{l,j+1}\rangle$ *for $\phi_i = \mathrm{X}_\exists\phi_l$ or $\phi_i = \mathrm{X}_\forall\phi_l$;*

for $j = n-1$:

- $\gamma(c_{i,j}) = \langle id, c_{r,j}\rangle$ *for $\phi_i = \phi_l \,\mathrm{U}\, \phi_r$ or $\phi_i = \phi_l \,\mathrm{R}\, \phi_r$,*
- $\gamma(c_{i,j}) = \mathtt{0}$ *for $\phi_i = \mathrm{X}_\exists\phi_l$, and*
- $\gamma(c_{i,j}) = \mathtt{1}$ *for $\phi_i = \mathrm{X}_\forall\phi_l$;*

for $0 \le j < n$:

- $\gamma(c_{i,j}) = 1$ *for either $\phi_i = p$ and $p \in \rho_j$ or $\phi_i = \neg p$ and $p \notin \rho_j$, $p \in P$,*
- $\gamma(c_{i,j}) = 0$ *for either $\phi_i = p$ and $p \notin \rho_j$ or $\phi_i = \neg p$ and $p \in \rho_j$, $p \in P$,*
- $\gamma(c_{i,j}) = \langle and, c_{l,j}, c_{r,j}\rangle$ *for $\phi_i = \phi_l \wedge \phi_r$, and*
- $\gamma(c_{i,j}) = \langle or, c_{l,j}, c_{r,j}\rangle$ *for $\phi_i = \phi_l \vee \phi_r$.*

Lemma 1. *The size of $C(\phi,\rho)$ is polynomial in $\|\rho\|$ and $\|\phi\|$. Moreover, in the evaluation of $C(\phi,\rho)$ the gate $c_{0,0}$ is labeled with the constant $\mathtt{1}$ if and only if $\rho \models \phi$.* □

In the remainder of the paper, we fix the formula ϕ and the path ρ, and refer to the circuit $C(\phi,\rho)$ as C. We now provide an embedding of C.

The *embedding* $Emb_C\colon \mathrm{gr}(C) \to 2^{\mathbb{R}\times\mathbb{R}}$ is defined by $Emb_C(c_{i,j}) = \{\langle j, \mathrm{depth}(\phi_i)\rangle\}$ and $Emb_C(c'_{i,j}) = \{\langle j+0.5, \mathrm{depth}(\phi_i)\rangle\}$, where $\mathrm{depth}(\phi_i)$ denotes the nesting depth of ϕ_i in ϕ. An edge of $\mathrm{gr}(C)$ is embedded to the line segment between the points onto which the incident nodes are embedded.

In general, Emb_C is not planar. However, for each subformula ϕ_i, $i = 1,\dots m-1$, we can identify a planar subcircuit $\mu_i = \bigcup_{j=0,\dots,n-1} C_{i,j}$, which we call the *module* of ϕ_i. Corresponding to the formula structure, the modules form a *module tree* $\mathcal{M} = \langle M, E\rangle$, where $M = \{\mu_i \mid i = 0,\dots,m-1\}$ and

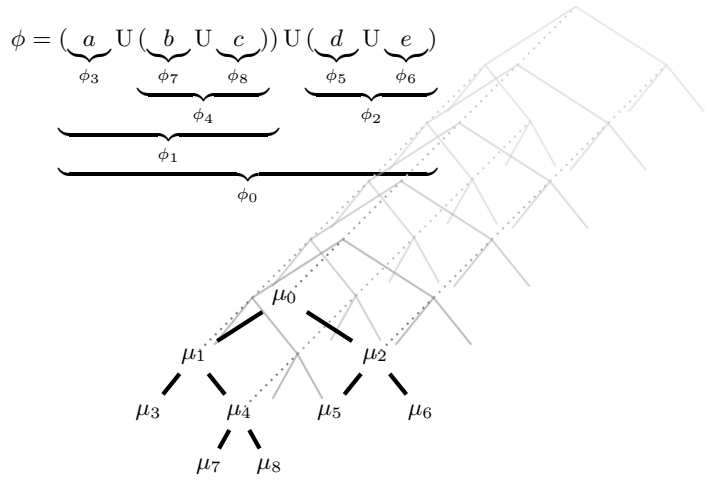

Fig. 2. A schematic illustration of the circuit and the module tree for a formula ϕ and a path of length six.

$E = \{\langle \mu_i, \mu_j \rangle \mid ((\phi_i = \phi_k \wedge \phi_l \text{ or } \phi_i = \phi_k \vee \phi_l \text{ or } \phi_i = \phi_k \,\mathrm{U}\, \phi_l \text{ or } \phi_i = \phi_k \,\mathrm{R}\, \phi_l) \text{ and } (j = l \text{ or } j = k)) \text{ or } \phi_i = \mathrm{X}_\exists \phi_j \text{ or } \phi_i = \mathrm{X}_\forall \phi_j\}$. Note that the modules are pairwise disjoint. A schematic illustration of an example circuit and the module tree is shown in Figure 2.

Figure 3 shows the partial circuit that corresponds to a single branch of the module tree from the example of Figure 2.

Our evaluation algorithm, which will be presented in the following section, uses the fast evaluation of OIF circuits from Corollary 1 to evaluate subcircuits of C. The following lemma establishes the connection between the embedding Emb_C and the module tree $\mathcal{M}$ that will allow for the application of Corollary 1 to increasingly larger subtrees of $\mathcal{M}$.

Lemma 2. *For a directed path $\pi \subseteq M$ in the tree $\mathcal{M}$, the circuit $P = \bigcup_{m \in \pi} m$ is planar. If P is* EV *and all variable gates in P belong to the terminating module of π then P is* OIF*. This property is stable under evaluation of partial circuits of C.*

Proof. The first sentence follows directly from the definition of Emb_C. The second sentence follows from the definition of Emb_C and the observation that an EV circuit with all variables on the outer face is OIF. For the proof of the third sentence, note that evaluation does not add any edge to $\mathrm{gr}(C)$. If the graph of P was planar (OIF) before the evaluation, it is planar (OIF) after the evaluation of any partial circuit of C. □

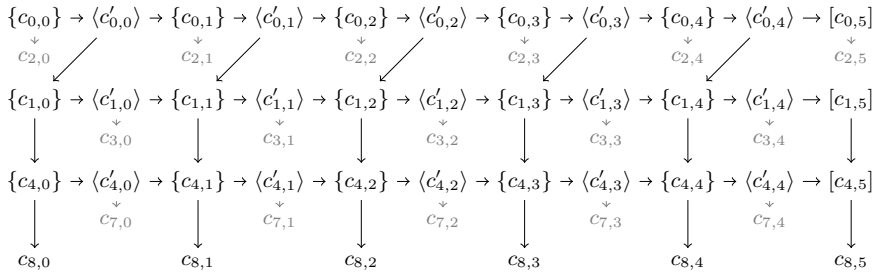

Fig. 3. The partial circuit for the modules $\mu_0, \mu_1, \mu_4, \mu_8$ from the example in Figure 2. The circuit is planar because the modules form a directed path in $\mathcal{M}$. Braces denote *or*-gates, angle brackets denote *and*-gates, and square brackets denote *id*-gates. Variable gates are shown in gray. For constant gates the labeling is omitted.

3 The Evaluation Algorithm

We now present our circuit evaluation algorithm. The problem of evaluating the circuit C from Section 2 is AC^1 Turing reduced to the evaluation problem for OIF circuits. Our algorithm repeatedly evaluates subcircuits of C. In the following, we always refer to the current circuit as C.

The central data structure of our algorithm is the *evaluation tree* $\mathcal{M}_{\simeq}$, which is the quotient of $\mathcal{M}$ with respect to an equivalence $\simeq$. As the algorithm progresses, more and more of the modules are collected into single nodes of $\mathcal{M}_{\simeq}$.

We define $\simeq$ as an equivalence relation on the modules of $\mathcal{M}$ such that the equivalence classes of $\simeq$ are full subtrees, i.e., for each equivalence class τ and each node $t \in \tau$, either each child or no child of t in $\mathcal{M}$ is in τ. For a node ν of $\mathcal{M}_{\simeq}$ we denote the circuit $\bigcup_{m \in \nu} m$ by $\mathrm{cir}(\nu)$. We call the nodes of $\mathcal{M}_{\simeq}$ the *e*nodes. An *e*node ν is called constant if $\mathrm{cir}(\nu)$ is constant.

Initially, each simple path in $\mathcal{M}$ forms a class. Starting from the leaves of $\mathcal{M}_{\simeq}$, our algorithm then evaluates the circuits corresponding to adjacent *e*nodes and updates $\simeq$ by collapsing the equivalence classes.

Throughout this process, we maintain the invariant that, for every *e*node, the corresponding partial circuit is OIF. This allows us to apply the evaluation algorithm from Corollary 1 on the partial circuit and, hence, perform the contraction within $\mathsf{logDCFL}$. The process ends when $\mathcal{M}_{\simeq}$ has been contracted into a single class. At that point, C is fully evaluated.

To ensure the invariant, we maintain that the equivalence relation is *well-formed*, as specified in the following definition:

Definition 3. *The equivalence relation $\simeq$ is well-formed if for each* e*node α of $\mathcal{M}_{\simeq}$ it holds that*

- *cir(α) is* EV*,*
- *α is a full subtree of $\mathcal{M}$, and*

– *α is either a leaf or there is a single module $bo(\alpha) \in \alpha$ such that there are modules $b, c \in M$ with $b \neq c$, $b, c \notin \alpha$, and $\langle a, b\rangle \in E$ and $\langle a, c\rangle \in E$.*

Together with Lemma 2, well-formedness ensures that for each *e*node, the corresponding circuit is OIF.

Lemma 3. *Let $\simeq$ be well-formed and let α be an* e*node of $\mathcal{M}_{\simeq}$. It holds that*

– *$\mathcal{M}_{\simeq}$ is a full binary tree,*
– *the circuit $cir(\alpha)$ is* OIF*, and*
– *if α is a leaf in $\mathcal{M}_{\simeq}$ then α is constant.*

Proof. Since *e*nodes are full subtrees of $\mathcal{M}$, the modules b and c from Definition 3 belong to different *e*nodes. Each *e*node is therefore either a leaf or has exactly two child *e*nodes. Hence, $\mathcal{M}_{\simeq}$ is a full binary tree. The uniqueness of $\mathrm{bo}(\alpha)$ implies that all variable gates of $\mathrm{cir}(\alpha)$ belong to $\mathrm{bo}(\alpha)$. Since $\mathrm{cir}(\alpha)$ is EV, all but the ancestor *e*nodes of $\mathrm{bo}(\alpha)$ are constant. Hence, all non-constant modules in α are on the directed path from the root of α to $\mathrm{bo}(\alpha)$. Since $\mathrm{cir}(\alpha)$ is EV, we conclude, by Lemma 2, that $\mathrm{cir}(\alpha)$ is OIF. If α is a leaf in $\mathcal{M}_{\simeq}$, then $\mathrm{cir}(\alpha)$ has no variable gates. Then α is constant, because $\mathrm{cir}(\alpha)$ is EV. □

Initialization. Initially, $\simeq$ is set to be the reflexive, symmetric, and transitive closure of $\simeq'$, where $a \simeq' b$ iff $(a, b) \in E$ and there is no c different from b s.t. $(a, c) \in E$.

$\mathrm{X}_\exists$ and $\mathrm{X}_\forall$ operators in the LTL formula give rise to modules with only a single child in $\mathcal{M}$. The initialization of $\simeq$ via $\simeq'$ causes the corresponding simple paths in $\mathcal{M}$ to collapse, such that $\mathcal{M}_{\simeq}$ is a full binary tree. Note that all classes of $\simeq$ are subtrees of $\mathcal{M}$.

To ensure well-formedness, we evaluate (in parallel) all non-singleton *e*nodes that contain constants. These are exactly the *e*nodes that correspond to modules originating from $\mathrm{X}_\exists$ and $\mathrm{X}_\forall$ operators stacked upon a single constant module. From the definition of Emb_C it is clear that those nodes are OIF and thus the evaluation can be performed in parallel within logDCFL, by using Corollary 1.

Lemma 4. *After the initial evaluation, $\simeq$ is well-formed.* □

Tree contraction. Each contraction step combines a leaf *e*node of $\mathcal{M}_{\simeq}$ with its parent and its sibling into a single *e*node. Well-formedness is preserved by evaluating the circuit of the resulting *e*node.

Lemma 5. *Let $\mathcal{M}_{\simeq}$ be well-formed. Given an* e*node α of $\mathcal{M}_{\simeq}$ with child* e*nodes β and γ. Let β be a leaf* e*node. The evaluation of $cir(\alpha \cup \beta \cup \gamma)$ can be performed in* logDCFL*. Updating $\simeq$ such that $\alpha \simeq \beta \simeq \gamma$ preserves well-formedness of $\mathcal{M}_{\simeq}$.*

Proof. Let $\mathcal{A} = \alpha \cup \beta \cup \gamma$. $\mathrm{cir}(\alpha)$ is OIF and β is constant. Thus the circuit $\mathrm{cir}(\alpha \cup \beta \cup \mathrm{const}(\gamma))$ is OIF and can be evaluated in logDCFL. After the evaluation, since $\mathrm{cir}(\gamma)$ is EV, all constants in $\mathrm{cir}(\mathcal{A})$ are sinks, and, hence, $\mathrm{cir}(\mathcal{A})$ is EV. $\mathcal{M}_{\simeq}$

is a full binary tree. Thus the $e$nodes α, β, γ together form a full subtree in $\mathcal{M}_{\simeq}$. $\mathcal{M}_{\simeq}$ is the quotient of $\mathcal{M}$, and α, β, and γ are each full subtrees of $\mathcal{M}$. It follows that $\mathcal{A}$ is a full subtree in $\mathcal{M}$ as well. In the subtree $\mathcal{A}$ of $\mathcal{M}$, the module $\mathrm{bo}(\alpha)$ is an internal node. Since β is a leaf, $\mathrm{bo}(\beta)$ does not exist. If γ is a leaf, $\mathcal{A}$ also becomes a leaf. Otherwise, $\mathrm{bo}(\mathcal{A}) = \mathrm{bo}(\gamma)$. □

Since, as we show in the following lemma, we can a contract a constant portion of the $e$nodes in parallel, the time consumed for the full contraction is logarithmic in the size of $\mathcal{M}$.

Lemma 6. *The circuit $C(\phi, \rho)$ can be evaluated within $AC^1(\mathsf{logDCFL})$.*

Proof. First, number the $e$nodes of $\mathcal{M}_{\simeq}$ that have a child that is a leaf from left to right (using depth first search on the tree, starting with 1) in L. Then, on every odd-numbered $e$node, apply Lemma 5. Since the involved circuits are disjoint for all odd-numbered $e$nodes, all applications of Lemma 5 can be performed in parallel. This eliminates at least $\left\lceil \frac{(\|\mathcal{M}_{\simeq}\|+1)/2}{2} \right\rceil$ leaves from the tree resulting in a tree $\mathcal{M}'_{\simeq}$ with $\|\mathcal{M}'_{\simeq}\| \leq \lfloor 3/4 \|\mathcal{M}_{\simeq}\| \rfloor$. Iterating this procedure leads in $O(\log \|\mathcal{M}_{\simeq}\|)$ steps to a single leaf $e$node. At this point, $C(\phi, \rho)$ is fully evaluated. The whole procedure can be implemented as an AC^1 circuit with $\mathsf{logDCFL}$ oracle gates.

The reduction circuit operates in stages. Each stage is structured as follows: an L oracle gate that takes the current $\mathcal{M}_{\simeq}$ as input identifies the sets of $e$nodes to be contracted on the current stage and feeds these into $\mathsf{logDCFL}$ oracle gates that implement Lemma 5. The remaining $e$nodes are just copied. The output of the stage is the updated version of $\mathcal{M}_{\simeq}$. Since $\mathsf{L} \subseteq \mathsf{logDCFL}$, each stage is of constant depth. A logarithmic number of sequential stages is stacked upon an initialization step that consists of a single L oracle gate that initializes $\mathcal{M}_{\simeq}$ from ϕ and ρ and parallel $\mathsf{logDCFL}$ oracle gates that evaluate $e$nodes that initially are simple paths in $\mathcal{M}$. □

Applying the evaluation algorithm to the circuit defined in Definition 2, we obtain an $\mathsf{AC}^1(\mathsf{logDCFL})$ solution to the path checking problem.

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

Proof. Given an LTL formula ϕ and a path ρ. In L build the circuit $C(\phi, \rho)$. Apply Lemma 6. The value of $c_{0,0}$ is the result. □

4 Conclusions

We have presented a positive answer to the question whether LTL can be checked efficiently in parallel on finite paths. Our construction can, for example, be used in hardware-based monitors to reduce the time needed to evaluate a block of path positions from linear to just logarithmic.

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 [14], in logCFL for semi-extended regular expressions [20], and P-complete for star-free regular expressions and extended regular expressions [19]. Of particular interest is the comparison to the star-free regular expressions, since they have the same expressive power as LTL on finite paths [16]. With $\mathsf{AC}^1(\mathsf{logDCFL})$ vs. P, our result demonstrates a computational advantage for LTL.

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 algorithm by Chakraborty and Datta (cf. Theorem 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 logDCFL.

References

1. Roy Armoni, Dimitry Korchemny, Andreas Tiemeyer, Moshe Y. Vardi, and Yael Zbar. Deterministic dynamic monitors for linear-time assertions. In *Proc. Workshop on Formal Approaches to Testing and Runtime Verification 2006*, volume 4262 of *Lecture Notes in Computer Science*. Springer, 2006.
2. 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.
3. David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen, and Sven Skyum. On monotone planar circuits. In *Proceedings of the 14^{th} Annual IEEE Conference on Computational Complexity (COCO '99)*, pages 24–31, Washington, DC, USA, 1999. IEEE Computer Society.
4. Marc Boule and Zeljko Zilic. Automata-based assertion-checker synthesis of PSL properties. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 13(1), 2008.
5. Tanmoy Chakraborty and Samir Datta. One-input-face MPCVP is hard for L, but in LogDCFL. In S. Arun-Kumar and Naveen Garg, editors, *Proc. FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 57–68. Springer, 2006.
6. Anat Dahan, Daniel Geist, Leonid Gluhovsky, Dmitry Pidan, Gil Shapir, Yaron Wolfsthal, Lyes Benalycherif, Romain Kamdem, and Younes Lahbib. Combining system level modeling with assertion based verification. In *ISQED '05: Proceedings of the 6th International Symposium on Quality of Electronic Design*, pages 310–315, Washington, DC, USA, 2005. IEEE Computer Society.
7. Arthur L. Delcher and S. Rao Kosaraju. An NC algorithm for evaluating monotone planar circuits. *SIAM J. Comput.*, 24(2):369–375, 1995.
8. S. Demri and Ph. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Inf. Comput.*, 174(1):84–103, 2002.
9. Patrick W. Dymond and Stephen A. Cook. Complexity theory of parallel time and hardware. *Information and Computation*, 80(3):205–226, 1989.
10. Bernd Finkbeiner and Henny B. Sipma. Checking finite traces using alternating automata. *Formal Methods in System Design*, 24:101–127, 2004.

11. Leslie M. Goldschlager. A space efficient algorithm for the monotone planar circuit value problem. *Inf. Process. Lett.*, 10(1):25–27, 1980.
12. K. Havelund and G. Roşu. Efficient monitoring of safety properties. *Software Tools for Technology Transfer*, 2004.
13. IEEE Std 1850-2007. *Standard for Property Specification Language (PSL)*. IEEE, New York, 2007.
14. Tao Jiang and Bala Ravikumar. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters*, 40:25–31, 1991.
15. S. Rao Kosaraju. On parallel evaluation of classes of circuits. In Kesav V. Nori and C. E. Veni Madhavan, editors, *Proc. FSTTCS*, volume 472 of *Lecture Notes in Computer Science*, pages 232–237. Springer, 1990.
16. Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218, London, UK, 1985. Springer.
17. Nutan Limaye, Meena Mahajan, and Jayalal M. N. Sarma. Evaluating monotone circuits on cylinders, planes and tori. In Bruno Durand and Wolfgang Thomas, editors, *Proc. STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 660–671. Springer, 2006.
18. Nicolas Markey and Philippe Schnoebelen. Model checking a path (preliminary report). In *Proc. CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 2003.
19. Holger Petersen. Decision problems for generalized regular expressions. In *Proc. 2nd International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures, London (Ontario)*, pages 22–29, 2000.
20. Holger Petersen. The membership problem for regular expressions with intersection is complete in LOGCFL. In Helmut Alt and Afonso Ferreira, editors, *Proc. STACS*, volume 2285 of *Lecture Notes in Computer Science*, pages 513–522. Springer, 2002.
21. Honghua Yang. An NC algorithm for the general planar monotone circuit value problem. In *Proc. 3rd IEEE Symposium on Parallel and Distributed Processing*, pages 196–203, 1991.
22. Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. CAV*, volume 2404 of *Lecture Notes in Computer Science*. Springer, 2002.