LTL Hierarchies and Model Checking

Radek Pelánek
Department of Computer Science, Faculty of Informatics
Masaryk University Brno, Czech Republic
xpelanek@fi.muni.cz

Abstract. We propose a new hierarchy of LTL formulas based on alternations of Until and Release operators and show that it is more relevant to model checking then previously studied hierarchies. Moreover, we study practically used formulas and conclude that in most cases it is possible to use specialized algorithms which are more efficient then general algorithm for LTL model checking.

1 Introduction

Linear Temporal Logic (LTL) is a popular formalism used in formal verification, particularly in model checking. Despite the high theoretical complexity bound, LTL model checking is well-known to be PSPACE-complete, LTL is successfully used in practice.

This contrast led Demri and Schonoebelen [6] to ask: *"What makes LTL model checking feasible in practice?"*. They tried to answer this question by the study of natural hierarchies of LTL formulas and complexity of model checking of their restricted classes. Unfortunately, they found that the PSPACE lower bound hold even for classes at the bottom of the hierarchies.

Notwithstanding this negative result, LTL hierarchies are a very active research area [27, 15, 25, 11, 21]. However, the recent works address mainly expressivity problems. In this work we overview most hierarchies studied in the literature with the following question in mind: *"Have these hierarchies any relation to model checking?"* We find out that most of the previously studied hierarchies do not have much connection to model checking.

Our main contribution is the introduction of a new hierarchy of LTL formulas based on the alternation depth of Until and Release operators. We provide the relation of this hierarchy with previously studied safety-progress hierarchy [22, 4] and show that it is possible to use more efficient algorithms for verification of formulas from lower classes of the Until/Release hierarchy.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

Thus we claim that this hierarchy is more relevant to model checking than the previously studied hierarchies.

Finally, we study practically used formulas from "Specification Patterns System" [7]. We find out that many practically used formulas lie in the lower classes of the Until/Release hierarchy.

Due to the space limitations we do not go into technical details and provide only general overview of our research.

2 Preliminaries

Linear Temporal Logic. The set of LTL formulas is defined inductively starting from a countable set AP of atomic propositions, Boolean operators, and the temporal operators $\mathtt{X}$ (Next) and $\mathtt{U}$ (Until):

$$\Psi := a \mid \neg\Psi \mid \Psi \vee \Psi \mid \Psi \wedge \Psi \mid \mathtt{X}\,\Psi \mid \Psi\,\mathtt{U}\,\Psi$$

LTL formulas are interpreted in the standard way [9] on infinite words over the alphabet 2^{AP}. We adopt standard abbreviations $\mathtt{R}, \mathtt{F}, \mathtt{G}$ for temporal operators Release ($\alpha\,\mathtt{R}\,\beta \equiv \neg(\neg\alpha\,\mathtt{U}\,\neg\beta)$), Future ($\mathtt{F}\,\alpha \equiv \mathit{true}\mathtt{U}\alpha$), and Globally ($\mathtt{G}\,\alpha \equiv \mathit{false}\,\mathtt{R}\,\alpha$) respectively.

Automata. A Büchi automaton is a tuple $A = \langle \Sigma, Q, q_0, \delta, F\rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to 2^Q$ is a nondeterministic transition function, and $F \subseteq Q$ is a set of accepting states. We differentiate *general*, *weak*, and *terminal* automata according to the following restrictions posed on their transition functions:

- *general*: none restriction
- *weak*: there exists a partition of the set Q into components Q_i and an ordering $\leq$ on these sets, such that for each $q \in Q_i, p \in Q_j$, if $\exists a \in \Sigma : q \in \delta(p, a)$ then $Q_i \leq Q_j$. Moreover for each Q_i, $Q_i \cap F = \emptyset$ or $Q_i \subseteq F$.
- *terminal*: for each $q \in F, a \in \Sigma$ it holds $\delta(q, a) \neq \emptyset$ and $\delta(q, a) \subseteq F$.

Terminal and weak automata are jointly called *specialized* automata.

Model Checking. The LTL model checking problem is to decide for a given system S and an LTL formula φ whether $S \models \varphi$, i.e. whether each run of S satisfy φ. The standard approach to LTL model checking is to construct an automaton $A_{\neg\varphi}$ which accepts exactly words which are *not* models of φ and then test the product automaton $S \times A_{\neg\varphi}$ for non-emptiness.

3 LTL Hierarchies

The most classical classes of LTL formulas are obtained by restricting the use of temporal operators. Following the usual notation ([9, 24]) we let

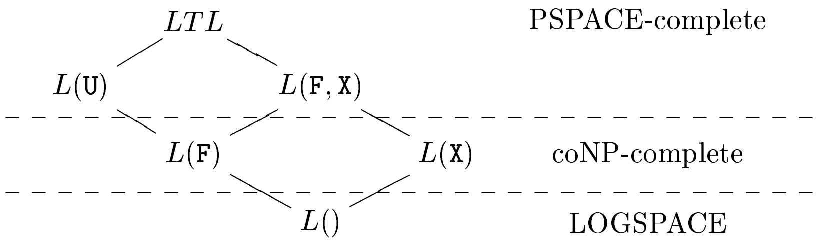

Figure 1.1: Inclusions which relate basic classes of LTL formulas into a hierarchy and complexity of their model checking.

$L(H_1, H_2, \ldots)$ denote the class of LTL for which only the temporal operators $H_1, H_2, \ldots$ are allowed. In this way we obtain six basic classes (see Fig. 1.1) – the classes obtained by combinations of other operators (such as $\mathtt{R}, \mathtt{G}$) coincide with one of these classes. Unfortunately, the complexity of model checking problem is very high even for restricted classes of the logic [24].

Practically important is the class $L(\mathtt{U})$. This class is closed under stuttering (i.e. it is insensitive to the number of successive occurrences of a letter in a word). Stutter-invariance enables to use efficient state space reduction techniques such as the partial order reduction or transition compression [17].

The hierarchy in Fig. 1.1 can be made more finer by constraining the maximum nesting depth of one or more operators. For example, we use the notation $L(\mathtt{U}^2, \mathtt{X})$ to denote the class of LTL with the use of modality $\mathtt{U}$ restricted to maximum nesting depth 2 and with the unrestricted use of modality $\mathtt{X}$. Researchers have studied several hierarchies based on the notion of the nesting depth, mainly $L(\mathtt{X}, \mathtt{F}, \mathtt{U}^m)$, $L(\mathtt{X}, \mathtt{U}^m)$, $L(\mathtt{X}^n, \mathtt{U})$, $L(\mathtt{X}^n, \mathtt{U}^m)$ [15, 11].

The work concerning these hierarchies is interested mainly with expressivity results: the strictness of hierarchies, characterization of languages in particular classes, decidability of membership in a given class, and connections to other formalisms (automata, semigroups, first order logic). The relations with model checking are sparse. Demri and Schnoebelen [6] have shown that the model checking problem is PSPACE-complete even for very restricted class such as $L(\mathtt{U}^2)$.

LTL can be extended with past operators like Since and Previous. The use of past operators do not add expressive power – classes that use past operators coincide with some of the pure-future classes. The advantage of using past operators is that the formulas can be exponentially more succinct than their pure-future counterparts [20] while the complexity of the model checking remains the same [23].

Another class is the consequence of a dispute concerning relative merits of linear and branching time logics (Vardi [26] gives a good overview). This dispute led to the study of relations among these logics [16] and to the search for properties expressible in both linear and branching time formalisms. The important class of this type is is the LTL^{det} class [21] – properties expressible in both LTL and ACTL. Model checking for this class can be solved in polynomial time.

Several other (more exotic) classes have been proposed. Dams suggested class based on a flat Until [5] – flat Until $\varphi_1 \mathtt{U}^f \varphi_2$ allows only propositional formula to appear in φ_1. However, the complexity of model checking remains the same for the flat class [6]. Another classes are for example these restricting the use of future operators in the scope of past operators [25, 23], or the number of atomic proposition [6]. None of these classes enable more efficient model checking.

Finally, we mention the membership problem. The problem is to decide, whether for a given formula φ and an LTL class C there is a formula φ' such that $\varphi \equiv \varphi'$ and $\varphi' \in C$. This problem is known to be decidable for most classes [27, 11, 15], but the complexity of the problem is usually very high (typically PSPACE-complete).

4 Until/Release Alternating Hierarchy

In this section we propose a new hierarchy of LTL formulas. This hierarchy is based on the *alternation* rather than the *nesting* of operators. We show the connection of the new hierarchy with previously studied safety-progress hierarchy and with model checking. Due to the space restrictions we state only main results without proofs.

Let us define hierarchies Σ_i^{LTL} and Π_i^{LTL} which reflect alternations of Until and Release operators in formulas. We use the Σ/Π notation since the way the hierarchy is defined strongly resembles the quantifier alternation hierarchy of first-order logic formulas or fixpoints alternation hierarchy of μ-calculus formulas [10].

Definition 1

The class $\Sigma_0^{LTL} = \Pi_0^{LTL}$ is the least set containing all atomic propositions and closed under the application of boolean and Next operators.

The class Σ_{i+1}^{LTL} is the least set containing Π_i^{LTL} and closed under the application of conjunction, disjunction, Next and Until operators.

The class Π_{i+1}^{LTL} is the least set containing Σ_i^{LTL} and closed under the application of conjunction, disjunction, Next and Release operators.

The class $\mathcal{B}_i^{LTL}$ is the boolean closure of Σ_i^{LTL} and Π_i^{LTL},

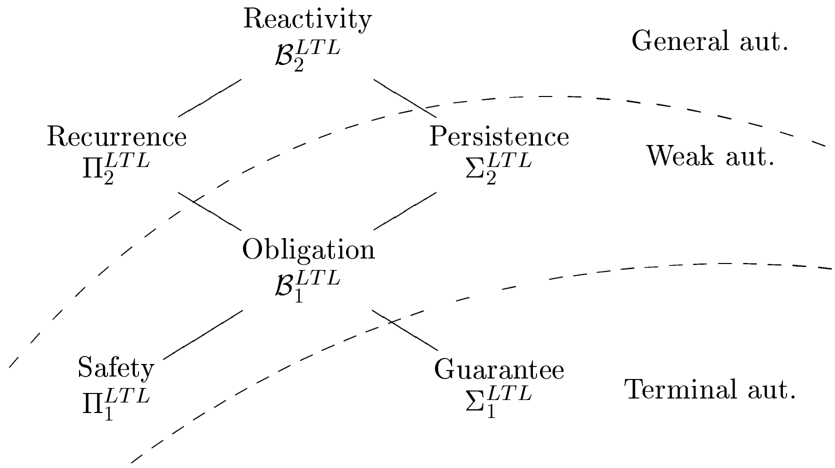

Figure 1.2: Safety-progress hierarchy, Until/Release hierarchy, and the corresponding types of Büchi automata

It shows up that our new hierarchy strongly relates to the safety-progress hierarchy defined by Manna and Pnueli [22, 4]. This is a classification of properties into a hierarchy consisting of six classes: *guarantee, safety, obligation, persistence, recurrence,* and *reactivity.* Inclusions, which relate the classes into a hierarchy, are depicted in Fig. 1.2. This hierarchy extends the safety-liveness characterization of ω-languages [18] and the Landweber hierarchy [19]. The classes of the hierarchy can be characterized through four views: a language-theoretic view, a topological view, a temporal logic view, and an automata view [22]. The fact that the hierarchy can be defined in many different ways shows the robustness of this hierarchy. Chang, Manna, and Pnueli have shown that the safety-progress hierarchy can be exploited for more efficient theorem proving [4]. We show that it can be used for better model checking as well.

The safety-progress hierarchy is practically orthogonal to the hierarchy from Fig. 1.1. The only remarkable relation to classes mentioned in the previous section is that $LTL^{det} \subseteq$ Recurrence. The relation with the Until/Release hierarchy is given by the following theorem:

Theorem 1 *A language that is specifiable by LTL is a guarantee (safety, obligation, persistence, recurrence, reactivity respectively) language if and only if it is specifiable by a formula from the class Σ_1^{LTL} ($\Pi_1^{LTL}, \mathcal{B}_1^{LTL}$, $\Sigma_2^{LTL}, \Pi_2^{LTL}, \mathcal{B}_2^{LTL}$ respectively) (see Fig. 1.2).*

Proof. [Sketch] The proof is based on the syntactic characterization of safety-progress classes by Chang, Manna, and Pnueli [4]. Using syntactic identities we are able to transform their characterization into the corresponding Σ^{LTL}/Π^{LTL} characterization.

As a consequence of the relation with safety-progress hierarchy we obtain the following, quite surprising fact (note that the previously mentioned hierarchies of first-order and μ-calculus formulas are infinite):

Theorem 2 *Both Σ_i^{LTL} and Π_i^{LTL} hierarchies semantically collapse – every LTL specifiable formula is specifiable by a $\mathcal{B}_2^{LTL}$ formula.*

The complexity of model checking is still PSPACE-complete even for lower classes of the hierarchy. Nevertheless, it is possible to employ this hierarchy for model checking. Formulas from lower classes of the hierarchy can be translated into specialized automata and the non-emptiness check for the product automaton can be performed more efficiently.

Theorem 3 *For every Σ_1^{LTL} (Σ_2^{LTL}) formula φ one can construct a terminal (weak) automaton accepting the language defined by φ.*

Proof. [Sketch] The basic idea of the construction is the same as for classical algorithm for transformation of LTL formula into automaton [13]. States of the automaton are sets of subformulas of the formula φ. The transition function is constructed in such a way that the following invariant is valid: if the automaton is in a state S then the remaining suffix of the word should satisfy all formulas in S. The main difference is in the way the acceptance condition is defined. For Σ_1^{LTL} and Σ_2^{LTL} formulas the acceptance condition can be simplified thanks to the special structure of alternation of Until and Release operators in the formula.

The non-emptiness of general automata is usually checked by nested depth-first search with explicit representation of the state space [14] or by nested fixpoint computation with quadratic number of symbolic steps with symbolic representation [12]. The non-emptiness of weak automata can be solved by single depth-first search [8] or by simple fixpoint computation with linear number of steps [2] respectively. The non-emptiness of terminal automata can be solved by classical reachability.

With the symbolic representation there is even asymptotic difference between the non-emptiness algorithms for general and specialized automata. All explicit algorithms have the same complexity, but the use of specialized algorithms still brings several benefits. Time and space optimization, "Guided search" heuristics [8], and the partial order reduction [14] can be employed more directly for specialized algorithms. Algorithms for specialized automata can be performed in distributed environment more easily [3]. Many of these benefits have been already demonstrated experimentally [2, 8, 3].

Since for Σ_1^{LTL} and Σ_2^{LTL} formulas we can use more efficient algorithms then for general formulas, the membership problem for these classes is of particular interest. This problem is decidable. Unfortunately, as for previously considered classes, the complexity of the problem is very high (the direct decision procedure goes via deterministic Streett automata and is doubly exponential). However, as we will see in the next section, practically used formulas are very short (thus it is feasible to perform expensive algorithms for them) and many of them even lie in Σ_1^{LTL} and Σ_2^{LTL} classes.

5 Practically Used Formulas

In order to find in which LTL classes practically used formulas lie, we have studied properties from the "Specification Patterns System" [7]. This system is a collection of the most often used model checking properties. The system provides properties in several different formalisms, LTL being one of them, together with their frequency (i.e. how often they are used) of each of them. We have observed following characteristics of these properties:

- The next operator is not used in these patterns, i.e. practically used properties are stutter-invariant.
- The most often used formulas are very short (three or less temporal operators).
- The nesting depth of until operator is in most cases one or two. The nesting depth is greater than three only in very rare cases.
- Most properties are either from safety class (41%) or from recurrence class (54%). This means that in most cases the resulting automaton is specialized[1].

The last observation is supported by Kupferman and Vardi [16] who claim that: "... most of the LTL formulas ψ used in practice are such that $A_{\neg\psi}$ is a 1-weak automaton" (1-weak automata are strict subclass of weak automata).

6 Conclusions and Future Work

The worst case complexity is not much useful with respect to the practical model checking. In practice, the computational complexity caused by LTL formula is "shadowed" by the size of a verified system. Thus it is important to use algorithms efficient with respect to a system. We conclude

[1] Please remember that in the verification process the negation of the formula is translated into an automaton. Therefore recurrence and safety formulas are translated into weak and terminal automata respectively.

that important classes of LTL are these which enable the use of such efficient algorithms — stutter-invariant class $L(\mathtt{U})$, which enable partial order reduction, and newly identified classes Σ_1^{LTL} (safety properties) and Σ_2^{LTL} (recurrence properties), which enable the use of specialized algorithms for non-emptiness check.

Our intended future works is to try to find out some new connections among other hierarchies and model checking. Particularly, we would like to study the connections with bounded model checking [1].

Acknowledgment

I thank Ivana Černá, my supervisor, for many valuable discussions and Jan Strejček for reading a draft of the paper.
The work on this paper has been partially supported by GA ČR grant number 201/03/0509.

Bibliography

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In W. R. Cleaveland, editor, *Proc. Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.

[2] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Proc. Computer Aided Verification*, volume 1633 of *LNCS*, pages 222–235. Springer, 1999.

[3] I. Černá and R. Pelánek. Distributed explicit fair cycle detection. In *Proc. SPIN workshop*, number 2648 in LNCS. Springer, 2003.

[4] E. Y. Chang, Z. Manna, and A. Pnueli. Characterization of temporal property classes. In *Proc. Automata, Languages and Programming*, volume 623 of *LNCS*, pages 474–486. Springer, 1992.

[5] D.R. Dams. Flat fragments of CTL and CTL*: Separating the expressive and distinguishing powers. *Logic Journal of the IGPL*, 7(1):55–78, 1999.

[6] S. Demri and Ph. Schnoebelen. The complexity of propositional linear temporal logics in simple cases (extended abstract). In *Proc. 15th Ann. Symp. Theoretical Aspects of Computer Science*, volume 1373, pages 61–72. Springer, 1998.

[7] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In *Proc. Workshop on Formal Methods in Software Practice*, pages 7–15. ACM Press, 1998.

[8] S. Edelkamp, A. L. Lafuente, and S. Leue. Directed explicit model checking with HSF-SPIN. In *Proc. SPIN workshop*, volume 2057 of *LNCS*, pages 57–79. Springer, 2001.

[9] E. A. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B, Formal Models and Semantics*, pages 995–1072. North-Holland Publishing Company, 1990.

[10] E. A. Emerson and C. L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proc. IEEE Symposium on Logic in Comuter Science*, pages 267 – 278. Computer Society Press, 1986.

[11] K. Etessami and T. Wilke. An Until hierarchy for temporal logic. In *Proc. IEEE Symposium on Logic in Computer Science*, pages 108–117. Computer Society Press, 1996.

[12] K. Fisler, R. Fraer, G. Kamhi Y. Vardi, and Z. Yang. Is there a best symbolic cycle-detection algorithm? In *Proc. Tools and Algorithms for Construction and Analysis of Systems*, volume 2031 of *LNCS*, pages 420–434. Springer, 2001.

[13] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.

[14] G. J. Holzmann, D. Peled, and M. Yannakakis. On nested depth first search. In *Proc. SPIN Workshop*, pages 23–32. American Mathematical Society, 1996.

[15] A. Kučera and J. Strejček. The stuttering principle revisited: On the expressiveness of nested X and U operators in the logic LTL. In *Proc. Computer Science Logic*, volume 2471 of *LNCS*, pages 276–291. Springer, 2002.

[16] O. Kupferman and M. Y. Vardi. Relating linear and branching model checking. In *Proc. Programming Concepts and Method*, pages 304–326. Chapman & Hall, 1998.

[17] R. P. Kurshan, V. Levin, and H. Yenigun. Compressing transitions for model checking. In *Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 569–581, 2002.

[18] L. Lamport. Proving correctness of multiprocess programs. *IEEE Transactions Software Engineering*, (3):125–143, 1977.

[19] L. H. Landweber. Decision problems for omega-automata. *Mathematical Systems Theory*, 3(4):376–384, 1969.

[20] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proc. Logic in Computer Science (LICS).* IEEE Computer Society, 2002.

[21] M. Maidl. The common fragment of CTL and LTL. In *Proc. 41th Annual Symposium on Foundations of Computer Science*, pages 643–652, 2000.

[22] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proc. ACM Symposium on Principles of Distributed Computing*, pages 377–410. ACM Press, 1990.

[23] N. Markey. Past is for free: on the complexity of verifying linear temporal properties with past. In *Proc. EXPRESS*, volume 68. Elsevier, 2002.

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

[25] D. Therien and T. Wilke. Nesting Until and Since in linear temporal logic. In *Proc. Symposium on Theoretical Aspects of Computer Science*, volume 2285 of *LNCS*, pages 455–464. Springer, 2002.

[26] M. Y. Vardi. Branching vs. linear time: Final showdown. In *Proc. Tools and Algorithms for Construction and Analysis of Systems*, volume 2031 of *LNCS*, pages 1–22. Springer, 2001.

[27] T. Wilke. Classifying discrete temporal properties. In *Symposium on Theoretical Aspects of Computer Science*, pages 32–46, 1999.