Thorough Checking Revisited

Shiva Nejati, Mihaela Gheorghiu, and Marsha Chechik
Department of Computer Science, University of Toronto,
Toronto, ON M5S 3G4, Canada.
Email:{shiva,mg,chechik}@cs.toronto.edu

***Abstract*— Recent years have seen a proliferation of 3-valued models for capturing abstractions of systems, since these enable verifying both universal and existential properties. Reasoning about such systems is either inexpensive and imprecise (compositional checking), or expensive and precise (thorough checking). In this paper, we prove that thorough and compositional checks for temporal formulas in their disjunctive forms coincide, which leads to an effective procedure for thorough checking of a variety of abstract models and the entire μ-calculus.**

I. Introduction

Recent years have seen a proliferation of approaches to capturing abstract models using rich formalisms that enable reasoning about arbitrary temporal properties. Examples of such formalisms include *Partial Kripke Structures (PKSs)* [1], *Mixed Transition Systems (MixTSs)* [2], [3], *Hyper-Transition Systems (HTSs)* [4], [5], [6], etc. Model checking over these is either conclusive, i.e., the property of interest can be proven or refuted, or inconclusive, denoted maybe, indicating that the abstract model needs to be refined.

Two distinct 3-valued semantics of temporal logic are used over these abstract models. One is *compositional*, in which the value of a property is computed from the values of its subproperties (as in classical model checking), and the other one is *thorough* [1]. The latter assigns maybe to a property only if there is a pair of concretizations of the abstract model such that the property holds in one and fails in the other. In general, model checking with thorough semantics is more expensive than compositional model checking – EXPTIME-complete for CTL, LTL and μ-calculus (L_μ) [7]. Thorough semantics, however, is more conclusive than compositional. For example, consider the program P shown in Figure 1(b), where `x` and `y` are integer variables and `x, y = e1, e2` indicates that `x` and `y` are simultaneously assigned `e1` and `e2`, respectively. A PKS M, shown in Figure 1(c), is an abstraction of P w.r.t. predicates p (meaning "`x` is odd"), and q, (meaning "`y` is odd"). The CTL formula $\varphi = AGq \wedge A[pU\neg q]$ evaluates to maybe on M under compositional semantics and to false under thorough, since every refinement of M refutes φ.

For the purpose of effective reasoning about abstract models, it is important to enable thorough-quality analysis using (compositional) 3-valued model checking. Specifically, we aim to identify classes of temporal formulas whose compositional model checking is as precise as thorough. Otherwise, we want to transform the formulas into equivalent ones (in classical logic), for which compositional checking yields the most precise answer. For example, we would transform the formula $AGq \wedge A[pU\neg q]$ into $A[p \wedge q\ U\ \mathsf{false}]$, that is unsatisfiable (over total models) and thus always false. [9], [8] refer to this process as *semantic minimization*, and the formulas for which thorough and compositional semantics coincide as *self-minimizing*.

This paper addresses thorough checking of L_μ formulas over various abstract models with 3-valued semantics following the algorithm in Figure 1(a). This algorithm consists of three main steps: (1) (compositional) model checking of φ over an abstract model M (e.g.,[1], [2], [5]), (2) checking if φ is self-minimizing, and (3) computing semantic minimization of φ, and then model checking the resulting formula. Computing semantic minimization is the most expensive part and is at least as hard as thorough checking [8]. Therefore, it is important to identify as many self-minimizing formulas as possible in step (2), and avoid step (3).

In [8] and [10], it was shown that positive/monotone temporal formulas, i.e., the ones that do not include both p and $\neg p$, are self-minimizing over abstract models described as PKSs. This self-minimization check, however, is not robust for more expressive abstraction modelling formalisms such as HTSs. For example, consider an abstraction of program P, described as an HTS H, shown in Figure 1(d). Based on the 3-valued semantics of L_μ over HTSs, the monotone formula $AGp \wedge AGq$ evaluates to maybe over H [5], [4]. However, this formula is false by thorough checking, because every concretization of H refutes either AGp or AGq.

In this paper, we extend step (2) of the thorough checking algorithm by proving that the disjunctive and conjunctive normal forms of L_μ defined in [11] are self-minimizing over abstract models described as HTSs. We focus on HTSs because other 3-valued abstraction formalisms can be translated to them without loss of precision, but not the other way around [4]. Godefroid and Huth [8] proved that *monotone* disjunctive L_μ formulas *without* greatest fixpoints are self-minimizing for PKSs, and pointed out that by a naive inductive proof the self-minimization of greatest fixpoint disjunctive formulas cannot be shown. We improve on this result by using an automata intersection game inspired by [12], to show that the disjunctive and conjunctive normal forms of L_μ *with* greatest fixpoint are self-minimizing over HTSs. Our result yields a simple syntactic check for identifying self-minimizing formulas over HTSs and can be used along with the monotonicity condition for PKSs and MixTSs.

Our result further provides an alternative semantic minimization procedure for step (3) of the algorithm, via the

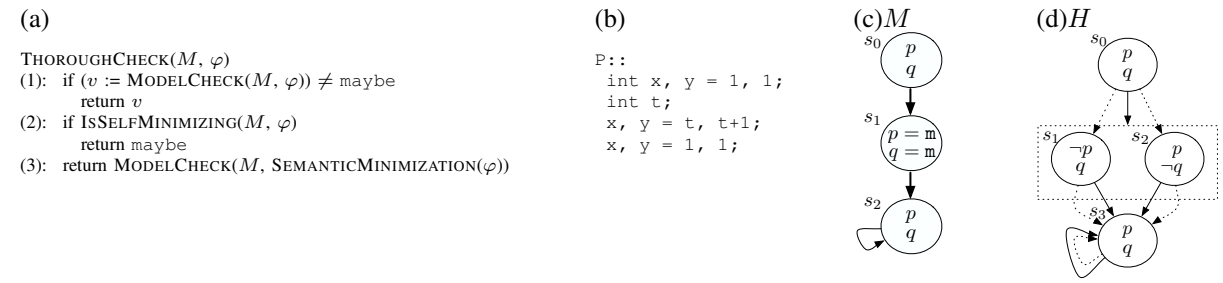

Fig. 1. (a) A sketch of an algorithm for thorough checking. A simple program P (adapted from [8]) (b) and its abstractions described as: (c) a PKS M; and (d) an HTS H.

tableau-based translation of Janin and Walukiewicz [11]. Godefroid and Huth [8] proved that L_μ formulas are closed under semantic minimization, i.e., every L_μ formula can be translated to an equivalent L_μ formula (in classical logic), for which compositional checking yields the most precise answer. The translation, however, is complicated and includes several steps: transforming L_μ formulas to non-deterministic tree automata, making non-deterministic tree automata 3-valued, and translating back these automata to L_μ. Our semantic minimization procedure is more straightforward and only uses the simple tableau-based construction described in [11]. Finally, we show that our semantic minimization procedure can be extended to abstract models described as PKSs and MixTSs, thus providing a general SemanticMinimization() subroutine for the algorithm in Figure 1(a).

The rest of this paper is organized as follows: Section II outlines some preliminaries. Section III defines an automata intersection game inspired by the abstraction framework in [12]. This game is used in Section IV to prove the main result of the paper which establishes a connection between self-minimizing formulas over HTSs and disjunctive/conjunctive forms of L_μ. Section V provides a complete algorithm for thorough checking of L_μ over arbitrary abstract models including PKSs, MixTSs, and HTSs, and discusses the complexity of this algorithm. In Section VI, we present some self-minimizing fragments of CTL for HTSs. We further discuss our work and compare it to related work in Section VII. Section VIII concludes the paper. Proofs for the major theorems are available in the extended version of this paper [13].

II. Preliminaries

In this section, we provide background on modelling formalisms, temporal logics, refinement relation, and compositional and thorough semantics.

3-valued logic. We denote by **3** the 3-valued Kleene logic [14] with elements true (t), false (f), and maybe (m). The truth ordering $\leq$ of this logic is defined as $\mathsf{f} \leq \mathsf{m} \leq \mathsf{t}$, and negation as $\neg\mathsf{t} = \mathsf{f}$ and $\neg\mathsf{m} = \mathsf{m}$. An additional ordering $\preceq$ relates values based on the amount of information: $\mathsf{m} \preceq \mathsf{t}$ and $\mathsf{m} \preceq \mathsf{f}$, so that m represents the least amount of information.

Models. In what follows, we introduce different modelling formalisms that are used in this paper.

A *Kripke structure (KS)* is a tuple $K = (\Sigma, s_0, R, L, AP)$, where Σ is a set of states, $s_0 \in \Sigma$ is the initial state, $R \subseteq \Sigma \times \Sigma$ is a transition relation, AP is the set of *atomic propositions*, and $L : \Sigma \to 2^{AP}$ is a labelling function. We assume KSs are total, i.e., R is left-total.

A *Partial Kripke Structure (PKS)* [1] is a KS whose labelling function L is 3-valued, i.e., $L : \Sigma \to \mathbf{3}^{AP}$. Figure 1(c) illustrates a PKS, where propositions p and q are m in state s_1.

An *Mixed Transition System (MixTS)* [2], [3] is a tuple $(\Sigma, s_0, R^{must}, R^{may}, L, AP)$, where Σ is a set of states, $s_0 \in \Sigma$ is the initial state, $R^{must} \subseteq \Sigma \times \Sigma$ and $R^{may} \subseteq \Sigma \times \Sigma$ are *must* and *may* transition relations, respectively, AP is the set of atomic propositions, and $L : \Sigma \to \mathbf{3}^{AP}$ is a 3-valued labelling function.

A *hyper-transition system (HTS)* [4], [5], [6] is a tuple $H = (\Sigma, s_0, R^{must}, R^{may}, L, AP)$, where $R^{must} \subseteq \Sigma \times \mathcal{P}(\Sigma)$ and $R^{may} \subseteq \Sigma \times \Sigma$ are *must* and *may* transition relations, respectively, $L : \Sigma \to 2^{AP}$ is a 2-valued labelling function, and Σ, s_0 and AP are defined as above. Intuitively, an HTS is a MixTS with a 2-valued labelling function and must hyper-transitions. We assume HTSs and MixTSs are total, i.e., R^{may} is left-total. Figure 1(d) illustrates an HTS, where *must* and *may* transitions are represented as solid and dashed arrows, respectively. Throughout this paper, we often write relations as functions: for instance, $R^{may}(s)$ is the set $\{s' \mid (s, s') \in R^{may}\}$.

An HTS H is *concrete* if for every $s, s' \in \Sigma$, we have $s' \in R^{may}(s) \Leftrightarrow \{s'\} \in R^{must}(s)$. For every KS $K = (\Sigma, s_0, R, L, AP)$, there is an equivalent concrete HTS $H_K = (\Sigma, s_0, R^{must}, R^{may}, L, AP)$, where $R^{may} = R$ and $s' \in R(s) \Leftrightarrow \{s'\} \in R^{must}(s)$ for every $s, s' \in \Sigma$.

Temporal logics. Temporal properties are specified in the *propositional μ-calculus* L_μ [15].

Definition 1: Let Var be a set of fixpoint variables, and AP be a set of atomic propositions. The *logic* $L_\mu(AP)$ is the set of formulas generated by the following grammar:

$$\varphi ::= \mathsf{true} \mid p \mid Z \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid EX\varphi \mid \mu Z \cdot \varphi(Z)$$

where $p \in AP$, $Z \in Var$, and $\varphi(Z)$ is syntactically monotone in Z.

The derived connectives are defined as follows:

$$\begin{array}{lcl} \varphi_1 \vee \varphi_2 & = & \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\ AX\varphi & = & \neg EX\neg\varphi \\ \nu Z \cdot \varphi(Z) & = & \neg\mu Z \cdot \neg\varphi(\neg Z) \end{array}$$

Any L_μ formula can be transformed into an equivalent formula in which negations are applied only to atomic propositions. Such formulas are said to be in negation normal form

$$
\begin{array}{lcl}
||\mathsf{true}||_\top e & = & \Sigma \\
||\mathsf{true}||_\bot e & = & \emptyset \\
||p||_\top e & = & \{s \mid p \in L(s)\} \\
||p||_\bot e & = & \{s \mid p \notin L(s)\} \\
||Z||_\top e & = & e(Z) \\
||Z||_\bot e & = & \overline{e(Z)} \\
||\neg\varphi||_\top e & = & ||\varphi||_\bot e \\
||\neg\varphi||_\bot e & = & ||\varphi||_\top e \\
||\varphi_1 \wedge \varphi_2||_\top e & = & ||\varphi_1||_\top e \cap ||\varphi_2||_\top e \\
||\varphi_1 \wedge \varphi_2||_\bot e & = & ||\varphi_1||_\bot e \cup ||\varphi_2||_\bot e \\
||EX\varphi||_\top e & = & ex(||\varphi||_\top e) \\
||EX\varphi||_\bot e & = & ax(||\varphi||_\bot e) \\
||\mu Z \cdot \varphi||_\top e & = & \bigcap\{S \subseteq \Sigma \mid ||\varphi||_\top e[Z \to S] \subseteq S\} \\
||\mu Z \cdot \varphi||_\bot e & = & \bigcup\{S \subseteq \Sigma \mid S \subseteq ||\varphi||_\bot e[Z \to S]\}
\end{array}
$$

Fig. 2. The semantics of L_μ.

(NNF). An L_μ formula φ is universal (resp. existential) if $\mathtt{NNF}(\varphi)$ does not contain any EX (resp. AX) operators. We write $\varphi_\forall$ (resp. $\varphi_\exists$) to denote a universal (resp. existential) formula, and write φ_{prop} when φ is a propositional formula, i.e., when φ consists only of literals, conjunctions and disjunctions.

Definition 2: [5] Let H be an HTS, φ be an L_μ formula, and $e : Var \to \mathcal{P}(\Sigma)$ be an environment. We denote by $||\varphi||^H_\top e$ the set of states in H that satisfy φ, and by $||\varphi||^H_\bot e$ the set of states in H that refute φ. The sets $||\varphi||_\top e$ and $||\varphi||_\bot e$ are defined in Figure 2, where $ex(S) = \{s \mid \exists S' \in R^{must}(s) \cdot S' \subseteq S\}$ and $ax(S) = \{s \mid \forall s' \in R^{may}(s) \cdot s' \in S\}$.

For a closed L_μ formula φ, $||\varphi||^H_\lambda e_1 = ||\varphi||^H_\lambda e_2$ for any e_1 and e_2 and $\lambda \in \{\top, \bot\}$. Thus, e can be safely dropped when φ is closed. We also omit H when it is clear from the context. Since KSs are special cases of HTSs, the above semantics applies to them as well.

In this paper, we often express temporal formulas in the *computation tree logic* CTL [16] whose syntax is defined w.r.t. a set AP of atomic propositions as follows:

$$
\begin{array}{rcl}
\varphi & ::= & p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid EX\varphi \mid AX\varphi \mid E[\varphi U \varphi] \mid \\
& & A[\varphi U \varphi] \mid E[\varphi \tilde{U} \varphi] \mid A[\varphi \tilde{U} \varphi]
\end{array}
$$

where $p \in AP$. The operators AU and EU are universal and existential until operators, respectively; and operators $E\tilde{U}$ and $A\tilde{U}$ are their duals, respectively. Other CTL operators can be defined from these:

$$
\begin{array}{lcllcl}
AG\varphi & = & A[\mathsf{false}\tilde{U}\varphi] & EG\varphi & = & E[\mathsf{false}\tilde{U}\varphi] \\
AF\varphi & = & A[\mathsf{true}U\varphi] & EF\varphi & = & E[\mathsf{true}U\varphi]
\end{array}
$$

CTL has a fixpoint characterization which provides a straightforward procedure for translating CTL to L_μ. Thus, the semantics of CTL over HTSs follows from Definition 2.

3-valued compositional semantics. An HTS H is *consistent* [4] if for every $s \in \Sigma$ and $S \in R^{must}(s)$, $S \cap R^{may}(s) \neq \emptyset$. Therefore, for every consistent HTS H and $\varphi \in L_\mu$, $||\varphi||_\top \cap ||\varphi||_\bot = \emptyset$, i.e., a consistent H does not satisfy $\varphi \wedge \neg\varphi$.

The semantics of L_μ over a consistent HTS H can be described as a 3-valued function $||.||^H_{\mathbf{3}} : L_\mu \times \Sigma \to \mathbf{3}$. We write $||\varphi||^H_{\mathbf{3}}(s) = \mathsf{t}$ if $s \in ||\varphi||^H_\top$, $||\varphi||^H_{\mathbf{3}}(s) = \mathsf{f}$ if $s \in ||\varphi||^H_\bot$, and $||\varphi||^H_{\mathbf{3}}(s) = \mathsf{m}$ otherwise. The value of φ in H, denoted $||\varphi||^H_{\mathbf{3}}$, is defined as $||\varphi||^H_{\mathbf{3}}(s_0)$, where s_0 is the initial state of H. To disambiguate from an alternative semantics presented later, we refer to this 3-valued semantics of L_μ over HTSs as *compositional*.

Refinement relation. Models with 3-valued semantics are compared using ordering relations known as *refinement relations* [17].

Definition 3: [5] Let H_1 and H_2 be HTSs. A *refinement relation* $\rho \subseteq \Sigma_1 \times \Sigma_2$ is the largest relation where $\rho(s,t)$ iff

1) $L_1(s) = L_2(t)$,
2) $\forall S \subseteq \Sigma_1 \cdot R_1^{must}(s,S) \Rightarrow \exists T \subseteq \Sigma_2 \cdot R_2^{must}(t,T) \wedge \hat{\rho}(S,T)$,
3) $\forall t' \in \Sigma_2 \cdot R_2^{may}(t,t') \Rightarrow \exists s' \in \Sigma_1 \cdot R_1^{may}(s,s') \wedge \rho(s',t')$,

where $\hat{\rho}(S,T) \Leftrightarrow \forall t' \in T \cdot \exists s' \in S \cdot \rho(s',t')$.

We say H_2 refines H_1 and write $H_1 \preceq H_2$, if there is a refinement ρ such that $\rho(s_0^1, s_0^2)$, where s_0^1 and s_0^2 are the initial states of H_1 and H_2, respectively.

Refinement preserves L_μ formulas [5], i.e., if $H_1 \preceq H_2$, then for every $\varphi \in L_\mu$, $||\varphi||^{H_1}_{\mathbf{3}} \preceq ||\varphi||^{H_2}_{\mathbf{3}}$. Refinement can relate HTSs to KSs as well. Recall that every KS K is equivalent to a concrete HTS H_K. We say that a KS K refines an HTS H, denoted $H \preceq K$, iff $H \preceq H_K$. For an HTS H, let $\mathcal{C}[H]$ denote the set of *completions* of H, that is, the set of all KSs that refine H.

Thorough semantics and semantic minimization. Thorough semantics of L_μ over HTSs is defined w.r.t. the completions of HTSs: A formula φ is true in H under thorough semantics, written $||\varphi||^H_t = \mathsf{t}$, if it is true in all completions of H; it is false in H, written $||\varphi||^H_t = \mathsf{f}$, if it is false in all completions of H, and it is maybe in H, written $||\varphi||^H_t = \mathsf{m}$, otherwise [1].

Thorough semantics is more precise than compositional semantics [1]. That is, $||\varphi||^H_{\mathbf{3}} \preceq ||\varphi||^H_t$ for every HTS H and L_μ formula φ. A formula φ is a *positive semantic minimization* of a formula φ' if for every HTS H, $||\varphi'||^H_t = \mathsf{t} \Leftrightarrow ||\varphi||^H_{\mathbf{3}} = \mathsf{t}$, and is a *negative semantic minimization* of φ' if for every HTS H, $||\varphi'||^H_t = \mathsf{f} \Leftrightarrow ||\varphi||^H_{\mathbf{3}} = \mathsf{f}$. Further, a formula φ is called *positively self-minimizing* when it is its own positive semantic minimization, and is *negatively self-minimizing* when it is its own negative semantic minimization. A formula that is both positively and negatively self-minimizing is called *semantically self-minimizing* or *self-minimizing* for short. For instance, $AGp \wedge AGq$ is not negatively self-minimizing, because for the HTS H in Figure 1(d), $||AGp \wedge AGq||^H_{\mathbf{3}} = \mathsf{m}$ and $||AGp \wedge AGq||^H_t = \mathsf{f}$. As we show later in the paper, $AG(p \wedge q)$ is a negative semantic minimization of $AGp \wedge AGq$. Dually, $EF(p \vee q)$ is a positive semantic minimization of $EFp \vee EFq$. Since thorough semantics is defined in terms of completions of HTSs, it is desirable to define self-minimizing formulas in the same terms, via an equivalent definition, as done below.

Definition 4: An L_μ formula φ is *negatively self-minimizing* if for every HTS H, $||\varphi||^H_{\mathbf{3}} \neq \mathsf{f} \Rightarrow \exists K \in \mathcal{C}[H] \cdot K \models \varphi$, and is *positively self-minimizing* if for every HTS H, $||\varphi||^H_{\mathbf{3}} \neq \mathsf{t} \Rightarrow \exists K \in \mathcal{C}[H] \cdot K \models \neg\varphi$.

Our definitions for positive and negative semantic minimization are, respectively, the same as those for *pessimistic* and *optimistic semantic minimization* in [8].

III. An Automata Intersection Game

In this section, we define an automata intersection game inspired by the automata-based abstraction framework proposed in [12]. In this framework, both temporal formulas and abstract

models are represented as finite automata. For a formula φ, the language of its corresponding automaton $\mathcal{A}_\varphi$ is the set of KSs satisfying φ, i.e., $K \in \mathcal{L}(\mathcal{A}_\varphi)$ iff $K \models \varphi$. For an abstract model H, the language of its corresponding automaton $\mathcal{A}_H$ is the set of completions of H, i.e., $\mathcal{C}[H] = \mathcal{L}(\mathcal{A}_H)$. Viewing formulas and abstract models as automata allows us to uniformly define both (thorough) model checking and refinement checking as automata language inclusion. That is, $H \models \varphi$ iff $\mathcal{L}(\mathcal{A}_H) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$ (model checking), and $H_1 \preceq H_2$ iff $\mathcal{L}(\mathcal{A}_{H_1}) \subseteq \mathcal{L}(\mathcal{A}_{H_2})$ (refinement checking) [12].

The class of automata used in [12] is known as μ-automata [11]. These automata, although very similar to non-deterministic tree automata (e.g., [18]), are more appropriate for working with branching time logics, because they precisely capture L_μ over transition systems with *unbounded* branching. We use a simplified definition of μ-automata adapted from [11], [12].

Definition 5: [11], [12] A *μ-automaton* is a tuple $\mathcal{A} = (Q, B, q_0, CH, BR, L, \Omega, AP)$, where Q is a non-empty, countable set of states called *choice* states; B is a countable set of states, disjoint from Q, called *branch* states; $q_0 \in Q$ is the initial state; $CH \subseteq Q \times B$ is a choice relation, from choice states to branch states; $BR \subseteq B \times Q$ is a transition relation, from branch states to choice states; $L : B \to 2^{AP}$ is a labelling function mapping each branch state to a subset of atomic propositions in AP; and $\Omega : Q \to \mathbb{N}$ is an indexing function, defining a parity acceptance condition.

Unless stated otherwise, "automata" and "μ-automata" are used interchangeably in the rest of the paper. Given an infinite tree T rooted at r_0, a *tree run* of an automaton $\mathcal{A}$ on T is an infinite tree T' whose root is labelled with (r_0, q_0). Every node of T' is labelled with either a pair (r, q) or (r, b), where r is a node from T, and q and b are respectively choice and branch states of $\mathcal{A}$. Every node (r, q) has at least one child node (r, b), where $b \in CH(q)$ and the labelling of b matches that of r. For every node (r, b) and every child r' of r in T, there exists a child (r', q') of (r, b) s.t. $q' \in BR(b)$. For every node (r, b) and every $q'' \in BR(b)$, there exists a child (r'', q'') of (r, b) s.t. r'' is a child of r in T. A *tree run T' is accepting* if on every infinite path π of T', the least value of $\Omega(q)$, for the choice states q that appear infinitely often on π, is even. An *input tree T is accepted* by $\mathcal{A}$ if there is *some* tree run of $\mathcal{A}$ on T that is accepting. The *language* of an automaton is the set of trees it accepts. For example, the language of the automaton shown in Figure 3(b) is the set of all infinite trees whose nodes are labelled by $\{p, q\}$ or $\{p, \neg q\}$. Input trees for μ-automata have arbitrary branching degrees and are not necessarily binary. For a more detailed treatment of μ-automata, reader can refer to [12]. We give a translation from HTSs to automata as follows.

Definition 6: Let $H = (\Sigma, s_0, R^{must}, R^{may}, L, AP)$ be an HTS. The automaton associated with H, $\mathcal{A}_H = (Q, B, q_0, CH, BR, L', \Omega, AP)$, has choice states $Q = \{q_i \mid s_i \in \Sigma\}$, branch states $B = \{b_{i,S} \mid s_i \in \Sigma, S \subseteq R^{may}(s_i)\}$, and the initial state q_0 that corresponds to s_0. The labelling of a branch state $b_{i,S}$ is the labelling of s_i in H, i.e., $L'(b_{i,S}) = L(s_i)$. The indexing function assigns 0 to every choice state, making all choice states accepting. The transition relations are: $CH = \{(q_i, b_{i,S}) \mid \forall S' \in R^{must}(s_i) \cdot S' \cap S \neq \emptyset\}$ and $BR = \{(b_{i,S}, q_j) \mid s_j \in S\}$.

For example, the translation $\mathcal{A}_H$ of the HTS H in Figure 1(d) is shown in Figure 3(a). For every abstract model H, $\mathcal{L}(\mathcal{A}_H)$ should be equal to $\mathcal{C}[H]$. For a consistent HTS H, all of whose completions are expressible as KSs, our translation in Definition 6 guarantees that $\mathcal{L}(\mathcal{A}_H) = \mathcal{C}[H]$.

Theorem 1: Let H be a consistent HTS with the additional requirement that for every $s \in \Sigma$ and every $S \in R^{must}(s)$, we have $S \subseteq R^{may}(s)$. Then, $\mathcal{L}(\mathcal{A}_H) = \mathcal{C}[H]$.

The proof of Theorem 1 is similar to that of Lemma 1 in [12].

In [12], a game-based simulation over automata has been defined as a sufficient condition for language inclusion, i.e., if $\mathcal{A}_1$ is simulated by $\mathcal{A}_2$, then $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$. We adapt the definition of automata simulation from [12] to define an automata intersection game. We prove that the existence of a winning strategy for this game is a sufficient condition for non-emptiness of $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$. Each play in an automata intersection game is a sequence of pairs of states of the same type. Here, being of the same type means that both states are either choice states or branch states. A pair of choice (resp. branch) states is called a *choice (resp. branch) configuration*. At a choice configuration (q_1, q_2), only Player I can move, choosing one branch state in $CH(q_1)$ and one in $CH(q_2)$ that match on labels. Player I's goal is to find a common path that is accepted by both $\mathcal{A}_1$ and $\mathcal{A}_2$. At a branch configuration (b_1, b_2), Player II moves first and chooses any side, b_1 or b_2, and any successor of that side. Player I has to respond with a successor of the other side. Intuitively, Player I wins the play if Player II cannot steer it onto a path which is not accepted by either of the automata.

Definition 7: Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be automata with initial states q_0^1 and q_0^2, respectively. A *(q_0^1, q_0^2)-game* is defined as follows:

1) (Initial) The initial configuration is (q_0^1, q_0^2)
2) (Choice) In a choice configuration $(q_1, q_2) \in Q_1 \times Q_2$, Player I chooses b_1 in $CH_1(q_1)$ and b_2 in $CH_2(q_2)$. The play continues from configuration (b_1, b_2).
3) (Branch) In a branch configuration $(b_1, b_2) \in B_1 \times B_2$, the play can proceed in one of the following ways:
   a) The play ends and is a win for Player I if $L_1(b_1) = L_2(b_2)$; it is a win for Player II otherwise.
   b) Player II chooses a 'side' $i \in \{1, 2\}$, and a choice state q_i in $BR_i(b_i)$; Player I must respond with a choice state q_j in $BR_j(b_j)$ from the other side j. The play continues from the configuration (q_1, q_2).

If a finite play ends by rule 3a, the winner is as specified in that rule[1]. For an infinite play π and $i \in \{1, 2\}$, let $proj_i(\pi)$ be the infinite sequence from Q_i^ω obtained by projecting the choice configurations of π onto component i. Then, π is a win

[1]Since KSs are assumed to be total, we do not deal with finite plays in this paper. Thus, condition 3a in Definition 7 only ensures that if an infinite play π is won by Player I, then for every branch configuration (b_1, b_2) on π, $L_1(b_1) = L_2(b_2)$.

for Player I iff $proj_1(\pi)$ and $proj_2(\pi)$ satisfy the acceptance conditions for $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

We say that there is an *intersection relation* $\sqcap$ between $\mathcal{A}_1$ and $\mathcal{A}_2$, written as $\mathcal{A}_1 \sqcap \mathcal{A}_2$, if Player I has a winning strategy for the (q_0^1, q_0^2)-game.

Theorem 2: $\mathcal{A}_1 \sqcap \mathcal{A}_2$ implies $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) \neq \emptyset$.

By Definition 4, a formula φ is negatively self-minimizing if for every HTS H over which φ is non-false, there is a completion satisfying φ. In automata-theoretic terms, some completion of H satisfying φ exists iff $\mathcal{L}(\mathcal{A}_H) \cap \mathcal{L}(\mathcal{A}_\varphi) \neq \emptyset$. The following theorem shows that finding a winning strategy for the intersection game between an HTS automaton and a formula automaton is sufficient to show self-minimization.

Theorem 3: An L_μ formula φ is negatively self-minimizing if for every HTS H, $||\varphi||_{\mathbf{3}}^H \neq \mathsf{f} \Rightarrow \mathcal{A}_H \sqcap \mathcal{A}_\varphi$, and is positively self-minimizing if for every HTS H, $||\varphi||_{\mathbf{3}}^H \neq \mathsf{t} \Rightarrow \mathcal{A}_H \sqcap \mathcal{A}_{\neg\varphi}$.

This theorem follows from Theorem 2 and Definition 4. We use it in the next section to prove the main result of the paper.

IV. Disjunctive/Conjunctive L_μ and Self-Minimization

In this section, we introduce disjunctive L_μ and its dual, conjunctive L_μ, defined in [11], and prove that L_μ formulas in disjunctive and conjunctive forms are, respectively, negatively and positively self-minimizing.

We start by noting that arbitrary L_μ formulas may not be self-minimizing. For instance, HTS H in Figure 1(d) has completions that satisfy either AGp or AGq, but there is no completion satisfying $AGp \wedge AGq$. Thus, we cannot inductively prove that formulas of the form $\varphi_1 \wedge \varphi_2$ are negatively self-minimizing (or $\varphi_1 \vee \varphi_2$ positively self-minimizing). Intuitively, this is the same reason why *satisfiability* of $\varphi_1 \wedge \varphi_2$ cannot be proven by structural induction. [11] proposes a syntactic form of L_μ formulas, referred to as *disjunctive* L_μ, for which satisfiability can be proven inductively. The analogy between identifying negatively self-minimizing formulas and the satisfiability problem suggests that disjunctive L_μ may be negatively self-minimizing. We prove this below.

Definition 8: [11] Let Γ be a finite set of L_μ formulas. We define $\mathtt{ref}(\Gamma) = \bigwedge_{\psi\in\Gamma} EX\psi \wedge AX \bigvee_{\psi\in\Gamma} \psi$. *Disjunctive* L_μ, denoted $L_\mu^\vee$, is the set of formulas generated by the following grammar:

$$\varphi ::= p \mid \neg p \mid Z \mid \varphi \vee \varphi \mid \varphi_1 \wedge \ldots \wedge \varphi_n \mid \sigma(Z) \cdot \varphi(Z)$$

where $p \in AP$, $Z \in Var$ and $\sigma \in \{\mu, \nu\}$; and for $\sigma(Z)\cdot\varphi(Z)$, Z occurs in $\varphi(Z)$ only positively, and does not occur in any context $Z \wedge \psi$ or $\psi \wedge Z$ for some ψ; $\varphi_1 \wedge \ldots \wedge \varphi_n$ $(n > 1)$ is a *special conjunction*: every φ_i is either a literal (p or $\neg p$) or a formula of a form $\mathtt{ref}(\Gamma)$ for a finite set Γ of $L_\mu^\vee$ formulas, and at most one of the φ_i is of the form $\mathtt{ref}(\Gamma)$. Dually, we define *conjunctive* L_μ, denoted $L_\mu^\wedge$, consisting of all L_μ formula φ where $\mathtt{NNF}(\neg\varphi) \in L_\mu^\vee$.

Every $\varphi \in L_\mu^\vee$ can be linearly translated to a μ-automaton $\mathcal{A}_\varphi$ so that $K \in \mathcal{L}(\mathcal{A}_\varphi)$ iff $K \models \varphi$ [11]. For example, a μ-automaton $\mathcal{A}_{AGp}$ corresponding to AGp is shown in Figure 3(b) [2]. Let K be a KS over $AP = \{p, q\}$. K satisfies AGp iff all its states are labelled with $\{p, q\}$ or $\{p, \neg q\}$, i.e., unfolding K from its initial state results in an infinite tree, all of whose nodes, are labelled with $\{p, q\}$ or $\{p, \neg q\}$. Hence, the tree is accepted by $\mathcal{A}_{AGp}$, and so is K.

The formula $AG(p \wedge q) = \nu Z \cdot p \wedge q \wedge AXZ$ is in $L_\mu^\vee$, but $AGp \wedge AGq$ is not, because the conjunction is not special. A non-disjunctive formula such as $AGp \wedge AGq$ would be first written in its disjunctive form, $AG(p \wedge q)$, and then translated to a μ-automaton. Automaton $\mathcal{A}_{AG(p\wedge q)}$ is exactly the same as $\mathcal{A}_{AGp}$ but without branch state b_1.

Theorem 4: Every closed $L_\mu^\vee$ formula is negatively self-minimizing.

Using Definition 4, we can show by structural induction that every $L_\mu^\vee$ formula except the greatest fixpoint is negatively self-minimizing [8]. As argued in [8], a naive proof does not work for the greatest fixpoint formulas: Let $\varphi' = \nu Z \cdot \varphi(Z)$ and $||\varphi'||_{\mathbf{3}}^H \neq \mathsf{f}$. By the semantics of the greatest fixpoint, $||\varphi^i(\mathsf{t})||_{\mathbf{3}}^H \neq \mathsf{f}$ for every $i > 0$. By inductive hypothesis, for every i there is a completion K_i of H that satisfies $\varphi^i(\mathsf{t})$. While the sequence of $\varphi^i(\mathsf{t})$ converges to the fixpoint φ' on H, it is not clear whether the sequence of K_i converges to a completion of H satisfying φ'.

In our proof, we use the automata intersection game introduced in Section III. Instead of explicitly constructing a KS $K \in \mathcal{C}[H]$ satisfying φ', we prove that such a completion exists by showing a winning strategy of Player I for the game $\mathcal{A}_H \sqcap \mathcal{A}_{\varphi'}$. We sketch the proof and illustrate it by an example that uses a greatest fixpoint operator.

By inductive hypothesis, Player I has a winning strategy T^i for $\mathcal{A}_H \sqcap \mathcal{A}_{\varphi^i(\mathsf{t})}$ for every i. For a large enough i, we can convert T^i to a winning strategy T for $\mathcal{A}_H \sqcap \mathcal{A}_{\varphi'}$: Automaton $\mathcal{A}_{\varphi^i(\mathsf{t})}$ is a finite chain of unfoldings of $\mathcal{A}_{\varphi'}$, i.e., there is a morphism h which partially maps states of $\mathcal{A}_{\varphi^i(\mathsf{t})}$ to those of $\mathcal{A}_{\varphi'}$. We apply h to T^i to obtain T.

For example, let $\varphi' = AGp = \nu Z \cdot p \wedge AXZ$. For this formula, $\varphi(Z) = p \wedge AXZ$. Consider automata $\mathcal{A}_{AGp}$ and $\mathcal{A}_{\varphi^4(\mathsf{t})}$ shown in Figure 3(b) and (c) respectively. The mapping h is defined as follows: choice state $q_{\varphi^4(\mathsf{t})}$ is mapped to q_{AGp}, choice states $q_{\varphi^i(\mathsf{t})}$ to q_Z for $1 \leq i \leq 3$, and branch states $b_{l,i}$ to b_l for $l \in \{0, 1\}$ and $1 \leq i \leq 4$. State q_t and its corresponding branch states are left unmapped.

The winning strategy T^i is a function that for a choice configuration, returns a branch configuration, and for a branch configuration and a choice of Player II from either side, returns a choice state from the opposite side. We call two choice (branch) configurations (s_1, s_2) and (s_1, s_2') *indistinguishable by* h if $h(s_2) = h(s_2')$. For convenience, we extend h to pairs (s_1, s_2) of states, where s_1 is in $\mathcal{A}_H$ and s_2 is in $\mathcal{A}_{\varphi^i(\mathsf{t})}$ by letting $h(s_1, s_2) = (s_1, h(s_2))$. We define a strategy function T for Player I in the game $\mathcal{A}_H \sqcap \mathcal{A}_{\varphi'}$ as follows:

(i) For every choice configuration (q_1, q_2), let $T(q_1, q_2) = h(T^i(q_1, q_2'))$ for some q_2' in $\mathcal{A}_{\varphi^i(\mathsf{t})}$,

[2] In contrast to [12], [11], we assume that KSs are total. Therefore, every branch state in $\mathcal{A}_{AGp}$ has at least one successor.

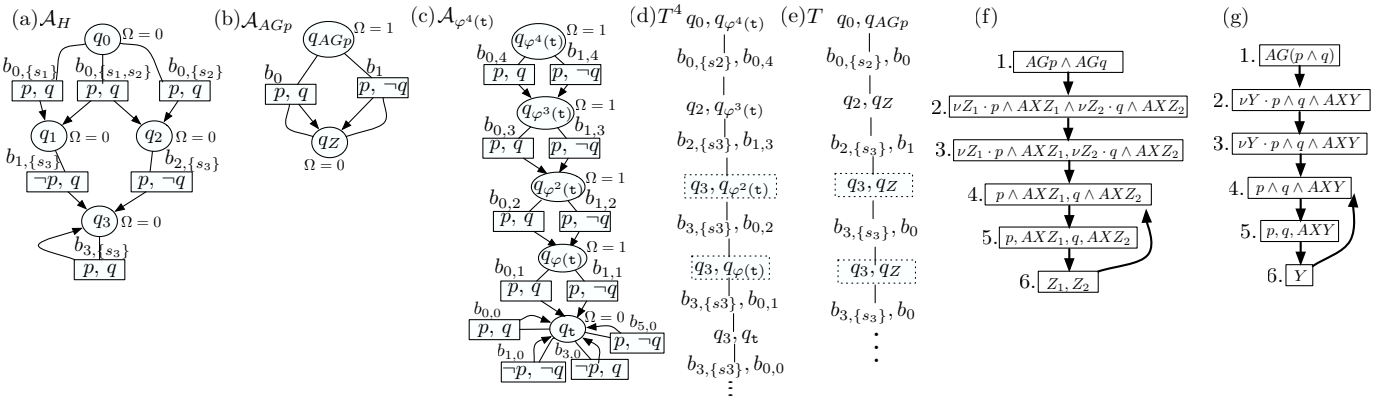

Fig. 3. Examples of automata, winning strategies, and semantic tableau: (a) automaton $\mathcal{A}_H$ corresponding to HTS H in Figure 1(d); (b) automaton $\mathcal{A}_{AGp}$; (c) automaton $\mathcal{A}_{\varphi^4(\mathrm{t})}$; (d) winning strategy T^4 for $\mathcal{A}_H \sqcap \mathcal{A}_{\varphi^4(\mathrm{t})}$; (e) winning strategy T for $\mathcal{A}_H \sqcap \mathcal{A}_{AGp}$; (f) the tableau associated to $AGp \wedge AGq$ [11]; and (g) extracting the disjunctive form of $AGp \wedge AGq$ from its tableau by the procedure of [11].

where $q_2 = h(q_2')$;

(ii) For every branch configuration (b_1, b_2) and every choice q_1 of Player II from $\mathcal{A}_H$, let $T(b_1, b_2, q_1) = h(T^i(b_1, b_2', q_1))$ for some b_2' in $\mathcal{A}_{\varphi^i(\mathrm{t})}$, where $b_2 = h(b_2')$;

(iii) For every branch configuration (b_1, b_2) and every choice q_2 of Player II from $\mathcal{A}_{\varphi'}$, let $T(b_1, b_2, q_2) = T^i(b_1, b_2', q_2')$ for some q_2' and b_2' in $\mathcal{A}_{\varphi^i(\mathrm{t})}$, where $q_2 = h(q_2')$ and $b_2 = h(b_2')$.

We first show that T is a function. Since unfoldings of $\mathcal{A}_{\varphi^i(\mathrm{t})}$ are isomorphic, there is some T^i s.t. for every two choice (branch) configurations indistinguishable by h, T^i returns configurations indistinguishable by h as well. That is, T^i returns the same results modulo h for arbitrary $b_2' \in h^{-1}(b_2)$ and $q_2' \in h^{-1}(q_2)$, making T a function.

Since h is undefined for q_{t}, to ensure that T is a valid strategy, we need to show that in the case (ii) above, there is always a b_2' where $T^i(b_1, b_2', q_1) \neq q_{\mathrm{t}}$. Since $\mathcal{A}_{\varphi'}$ and $\mathcal{A}_H$ are finite, such $b_2' \in h^{-1}(b_2)$ is found for a large enough i,

Strategy T as defined above is a valid strategy function for Player I in the game $\mathcal{A}_H \sqcap \mathcal{A}_{\varphi'}$. Note that T is undefined for the inputs for which T^i modulo h is undefined. It remains to show that T is winning. Let π be any play produced by T. Play π is an infinite sequence of configurations. Automata $\mathcal{A}_H$ and $\mathcal{A}_{\varphi'}$ are finite. Thus, there must be a configuration (q_1, q_2) repeated infinitely often in π. We map π back through h^{-1}, and distinguish two cases: (1) π is mapped back to a play π' generated by T^i s.t. infinitely many occurrences of (q_1, q_2) are mapped to a single configuration $(q_1, q_2' \neq q_{\mathrm{t}})$ in π'. Since T^i is a winning strategy for Player I, π' is won by Player I. Since h preserves parity of state indices, π is won by Player I in T as well. (2) π is mapped back to a prefix of some π' generated by T^i s.t. infinitely many pairs of consecutive occurrences of (q_1, q_2) are mapped to two different configurations (q_1, q_2') and (q_1, q_2'') in π', where $h(q_2') = h(q_2'')$. These two configurations are different; but since they are indistinguishable by h, they have to belong to two different unfoldings in the chain $\mathcal{A}_{\varphi^i(\mathrm{t})}$. Passing between unfoldings requires going through some state of the form $q_{\varphi^j(\mathrm{t})}$ for some $j < i$. Since $h(q_{\varphi^j(\mathrm{t})}) = q_Z$, there is an occurrence of (q, q_Z) for some state q of $\mathcal{A}_H$ between two consecutive occurrences of (q_1, q_2) in π. Thus, π satisfies the acceptance conditions of both $\mathcal{A}_H$ and $\mathcal{A}_{\varphi'}$ and as such, is a play won by Player I. This happens for every play generated by strategy T, and hence, strategy T is winning for Player I.

A winning strategy for $\mathcal{A}_H \sqcap \mathcal{A}_{\varphi^4(\mathrm{t})}$, denoted T^4, and its translation T by h are shown in Figures 3(d) and (e), respectively. For this example, unfolding automaton $\mathcal{A}_{\varphi(\mathrm{t})}$ four times is enough, because the only play produced by T^4 (shown in Figure 3(d)) visits two configurations $(q_3, q_{\varphi^2(\mathrm{t})})$ and $(q_3, q_{\varphi(\mathrm{t})})$ indistinguishable by h.

The following holds by duality to Theorem 4.

Theorem 5: Every closed $L_\mu^\wedge$ formula is positively self-minimizing.

Theorems 4 and 5 provide sufficient syntactic checks for identifying self-minimizing L_μ formulas that can be used in step (2) of the algorithm in Figure 1(a). Note that Theorems 4 and 5 only hold for HTSs, but not for PKSs or MixTSs. For example, $p \wedge \neg p$ is in $L_\mu^\vee$, but is not negatively self-minimizing over such models. Consider a model M with a single state in which proposition p is maybe. In M, $p \wedge \neg p$ is maybe, but this formula is false in any completion of M. In Section V, we show that by syntactically modifying disjunctive and conjunctive L_μ formulas, these formulas become negatively and positively self-minimizing over PKSs and MixTSs.

V. Thorough Checking Algorithm

In this section, we complete the thorough checking algorithm shown in Figure 1(a) by describing its subroutines IsSelfMinimizing() and SemanticMinimization(). Since we want this algorithm to work for arbitrary abstract models described as PKSs, MixTSs, or HTSs, we first need to show how disjunctive (resp. conjunctive) formulas can be made negatively (resp. positively) self-minimizing over these models.

Theorem 6: Let φ be a closed $L_\mu^\vee$ formula s.t. for every special conjunction $\psi = \psi_1 \wedge \ldots \wedge \psi_n$ in φ, there are no literals ψ_i and ψ_j $(1 \leq i, j \leq n)$ where $\psi_i = \neg\psi_j$. Then, φ is

```
THOROUGHCHECK(M, φ)
1:  if (v: = MODELCHECK(M, φ)) ≠ maybe
2:      return v
3:  if ISSELFMINIMIZING(M, φ)
4:      return maybe
5:  v := MODELCHECK(M, SEMANTICMINIMIZATION(φ))
6:  if (v = false) return false
7:  v := MODELCHECK(M, ¬(SEMANTICMINIMIZATION(NNF(¬φ))))
8:  if (v = true) return true
9:  return maybe

ISSELFMINIMIZING(M, φ)
10: if M is a PKS or an MixTS and φ is monotone
11:     return true
12: if M is an HTS and φ ∈ L_μ^∨ ∩ L_μ^∧
13:     return true
14: return false

SEMANTICMINIMIZATION(φ)
15: convert φ to its disjunctive form φ^∨
16: replace all special conjunctions in φ^∨ containing p and ¬p with false
17: return φ^∨
```

Fig. 4. The thorough checking algorithm.

negatively self-minimizing over abstract models described as HTSs, PKSs, or MixTSs.

The above theorem can be proven using the same argument as Theorem 4. The proof of Theorem 4 fails for MixTSs and PKSs, when some special conjunction in φ is of the form $p \wedge \neg p \wedge \ldots \wedge \varphi_n$, but Theorem 6 explicitly excludes this case, and hence, remains valid. Similarly, conjunctive formulas can be made positively self-minimizing for PKSs and MixTSs with a condition dual to that in Theorem 6.

The complete thorough checking algorithm is shown in Figure 4: THOROUGHCHECK() takes an abstract model M, described as an HTS, PKS or MixTS, and an L_μ formula φ, and returns the result of thorough checking φ over M. In THOROUGHCHECK(), semantic minimization is carried out in two steps: On line 5, φ is converted to its negative, and on line 7, to its positive semantic minimization formula. If model checking the negative semantic minimization returns false, φ is false by thorough checking, too; and if model checking the positive semantic minimization returns true, φ is true by thorough checking, as well.

If the model is a PKS or an MixTS, self-minimization follows from the monotonicity of φ, and so does the check in line 10 [8], [10]. Otherwise, we check whether $\varphi \in L_\mu^\vee \cap L_\mu^\wedge$ which, by our Theorems 4 and 5, guarantees self-minimization.

In SEMANTICMINIMIZATION(), φ is first converted to its disjunctive form $\varphi^\vee$ by the tableau-based conversion in [11]. Then, any special conjunction in φ containing two literals p and $\neg p$ is replaced with false. This ensures that $\varphi^\vee$ satisfies the condition in Theorem 6. Therefore, when passed φ (resp. $\mathtt{NNF}(\neg\varphi)$) as a parameter, SEMANTICMINIMIZATION() computes a negative (resp. positive) semantic minimization of φ.

To illustrate the algorithm, recall the formula $\varphi = AGq \wedge A[pU\neg q]$ from Section I. By compositional semantics, φ is maybe over both PKS M in Figure 1(c) and HTS H in Figure 1(d). Since φ is non-monotone and non-disjunctive, it is not self-minimizing for either M or H. SEMANTICMINIMIZATION() computes φ's negative semantic minimization by first converting it into a disjunctive form $\varphi^\vee = \mu Z \cdot (q \wedge AXAGq \wedge \neg q) \vee (p \wedge q \wedge AXZ)$, and then replacing the first conjunct with false. The result is the formula $\mu Z \cdot \mathsf{false} \vee (p \wedge q \wedge AXZ)$ which is false over both M and H, meaning that φ is false by thorough checking over both models. On the other hand, the formula $AGp \wedge AGq$ is monotone and thus self-minimizing for M. However, this formula is not disjunctive and thus not self-minimizing for H. SEMANTICMINIMIZATION() computes a negative semantic minimization of this formula by converting it to its disjunctive form $AG(p \wedge q)$ which turns out to be false over H. This shows that $AGp \wedge AGq$ is false by thorough checking over H.

Complexity. Let $\varphi \in L_\mu$ and M be an abstract model. The complexity of ISSELFMINIMIZING(M, φ) is linear in the size of φ, and that of MODELCHECK(M, φ) is $O((|\varphi| \cdot |M|)^{\lfloor d/2 \rfloor + 1})$, where d is the alternation depth of φ [19]. Thus, for the class of self-minimizing formulas, the running time of THOROUGHCHECK(M, φ) is the same as that of compositional model checking, i.e., $O((|\varphi| \cdot |M|)^{\lfloor d/2 \rfloor + 1})$.

The complexity of SEMANTICMINIMIZATION(φ), i.e., the complexity of converting an L_μ formula φ to its disjunctive $\varphi^\vee$ or conjunctive $\varphi^\wedge$ form, is $O(2^{O(|\varphi|)})$, producing formulas of size $O(2^{O(|\varphi|)})$ [11]. Therefore, for formulas requiring semantic minimization, the running time of THOROUGHCHECK(M, φ) is $O((2^{O(|\varphi|)} \cdot |M|)^{\lfloor d/2 \rfloor + 1})$, where d is the maximum of the alternation depths of $\varphi^\vee$ and $\varphi^\wedge$. When $\varphi^\vee$ and $\varphi^\wedge$ are alternation-free, i.e., $d = 0$, the complexity of THOROUGHCHECK(M, φ) becomes linear in the size of the abstract model, making the procedure efficient. However, we leave to future work the study of the relationships between the alternation depths of $\varphi^\vee$ and $\varphi^\wedge$ and that of φ.

VI. SELF-MINIMIZATION FOR CTL

In Section IV, we gave sufficient syntactic conditions for identifying self-minimizing L_μ formulas. Since CTL is used more often than L_μ in practice, it is useful to identify self-minimizing fragments of CTL as well. We do so by constructing grammars that generate positively/negatively self-minimizing CTL formulas.

[8] gives two grammars for negatively/positively self-minimizing formulas. Using our results on self-minimization checks of disjunctive/conjunctive L_μ, we extend these grammars as shown in Figure 5: φ^{neg} generates negatively and φ^{pos} generates positively self-minimizing formulas. The new constructs $A[\varphi_{prop} U \varphi^{neg}]$ and $A[\varphi^{neg} \tilde{U} \varphi_{prop}]$ added to the φ^{neg} grammar include formulas such as AGp and $A[pUq]$ that are negatively self-minimizing by Theorem 4. The construct $E[\varphi^{pos} U \varphi_{prop}]$ added to the φ^{pos} grammar includes, for instance, EFp that is positively self-minimizing by Theorem 5. Clearly, these grammars still do not capture the entire CTL which is not surprising because CTL is not closed under semantic minimization [8].

The notion of self-minimization in our grammars works only for HTSs. For example, φ^{neg} can generate $p \wedge \neg p$ which is not positively self-minimizing for either PKSs or MixTSs. To extend our grammars to these formalisms, we could restrict the grammar rules as in [9], [8] for propositional formulas, so that they do not produce non-monotone formulas.

$$\varphi^{neg} ::= p \mid \neg p \mid \varphi^{neg} \vee \varphi^{neg} \mid \varphi^{neg}_{\exists} \wedge \varphi^{neg}_{\exists} \mid \texttt{ref}(\Gamma^{neg}) \mid E[\varphi^{neg}_{\exists} U \varphi^{neg}] \mid E[\varphi^{neg}_{\exists} \tilde{U} \varphi^{neg}_{\exists}] \mid A[\varphi_{prop} U \varphi^{neg}] \mid A[\varphi^{neg} \tilde{U} \varphi_{prop}]$$

$$\varphi^{pos} ::= p \mid \neg p \mid \varphi^{pos} \wedge \varphi^{pos} \mid \varphi^{pos}_{\forall} \vee \varphi^{pos}_{\forall} \mid \widehat{\texttt{ref}}(\Gamma^{pos}) \mid E[\varphi^{pos} U \varphi_{prop}] \mid E[\varphi_{prop} \tilde{U} \varphi^{pos}] \mid A[\varphi^{pos}_{\forall} U \varphi^{pos}_{\forall}] \mid A[\varphi^{pos}_{\forall} \tilde{U} \varphi^{pos}]$$

Fig. 5. The grammar φ^{neg} (resp. φ^{pos}) generates negatively (resp. positively) self-minimizing subsets of CTL: Γ^{neg} (resp. Γ^{pos}) is a finite set of formulas generated by φ^{neg} (resp. φ^{pos}), and $\widehat{\texttt{ref}}$ is the dual of the operator $\texttt{ref}$.

VII. Related Work and Discussion

The problem of thorough checking for propositional logic was considered by [9] which proposed an efficient BDD-based algorithm for semantic minimization of propositional formulas. [1], [7] studied complexities and lower bounds for thorough checking of various temporal logics. [10] proposed self-minimizing checks for CTL, and [8] extended those checks to L_μ, and further, studied semantic minimization of various temporal logics.

In [8], a series of conversions between tree-automata and L_μ formulas is used to show that a semantic minimization of an L_μ formula can be computed in exponential time. This approach is hard to implement because it uses non-deterministic tree automata whose states have *unbounded* arities. The method proposed in [11] for translating an L_μ formula to its disjunctive form has the same complexity but is easier to implement, because it uses μ-automata instead– in this kind of automata, the number of successors of each state can be obtained from the structure of the formula.

As an example, the process of transforming $AGp \wedge AGq$ into its disjunctive form $AG(p \wedge q)$ was illustrated in Figures 3(e) and (f). The tableau for $AGp \wedge AGq$, constructed based on the L_μ proof rules of [11], is shown in Figure 3(e). The disjunctive form of $AGp \wedge AGq$ is constructed by traversing this tableau from its leaves to the top and labelling each node with a formula according to the procedure of [20] (see Figure 3(f)). Similar tableau methods were used in [21] for automata-based model checking of formulas whose conjunctions are restricted to having at most one conjunct with fixpoint variables.

In our paper, we only considered HTSs with 2-valued labels. This is in contrast to the HTSs in [4], [5] where states have 3-valued labels. Following [12], HTSs with 3-valued labels can be translated to ours. If the resulting HTSs satisfy the conditions in Theorem 1, then our results apply to the more general HTSs of [4], [5] as well.

VIII. Conclusion and Future Work

In this paper, we proved that disjunctive L_μ and conjunctive L_μ are, respectively, negatively and positively self-minimizing over HTSs. We base our proof on an automata intersection game. Our results provide a simple syntactic check for identifying self-minimizing formulas. For such formulas, thorough checking is as cheap as compositional model checking. We also proposed an algorithm for semantic minimization of L_μ and showed that its complexity is linear in the size of abstract models for L_μ formulas with alternation-free disjunctive and conjunctive forms.

In [7], it was shown that the complexity of thorough checking for the class of *persistence properties* [22], i.e., properties recognizable by co-Buchi automata, is also linear in the size of the abstract model. Studying the relationships between persistence properties and L_μ formulas with alternation-free disjunctive and conjunctive forms is left for future work.

Dams and Namjoshi [12] envisioned that viewing abstract models as μ-automata can open up many important connections between abstraction and automata theory. We believe that our work establishes one such connection, paving the way for further research on automata-based approaches to abstraction.

Acknowledgment. We thank Mehrdad Sabetzadeh for his help in improving the presentation of this paper. We thank the anonymous referees for their useful comments. Financial support was provided by NSERC and MITACS.

References

[1] G. Bruns and P. Godefroid, "Generalized model checking: Reasoning about partial state spaces," in *CONCUR*, ser. LNCS, vol. 1877, 2000, pp. 168–182.
[2] D. Dams, R. Gerth, and O. Grumberg, "Abstract interpretation of reactive systems," *ACM TOPLAS*, vol. 2, no. 19, pp. 253–291, 1997.
[3] R. Cleaveland, S. P. Iyer, and D. Yankelevich, "Optimality in abstractions of model checking," in *SAS*, ser. LNCS, vol. 983, 1995, pp. 51–63.
[4] L. de Alfaro, P. Godefroid, and R. Jagadeesan, "Three-valued abstractions of games: Uncertainty, but with precision," in *LICS*, 2004, pp. 170–179.
[5] S. Shoham and O. Grumberg, "Monotonic abstraction-refinement for ctl," in *TACAS*, ser. LNCS, vol. 2988, 2004, pp. 546–560.
[6] K. Larsen and L. Xinxin, "Equation solving using modal transition systems," in *LICS*, 1990.
[7] P. Godefroid and R. Jagadeesan, "Automatic abstraction using generalized model-checking," in *CAV*, ser. LNCS, vol. 2404, 2002, pp. 137–150.
[8] P. Godefroid and M. Huth, "Model checking vs. generalized model checking: Semantic minimizations for temporal logics," in *LICS*, 2005, pp. 158–167.
[9] T. Reps, A. Loginov, and S. Sagiv, "Semantic minimization of 3-valued propositional formulae," in *LICS*, 2002.
[10] A. Gurfinkel and M. Chechik, "How thorough is thorough enough," in *CHARME*, ser. LNCS, vol. 3725, 2005, pp. 65–80.
[11] D. Janin and I. Walukiewicz, "Automata for the modal mu-calculus and related results," in *MFCS*, 1995, pp. 552–562.
[12] D. Dams and K. Namjoshi, "Automata as abstractions," in *VMCAI*, 2005, pp. 216–232.
[13] S. Nejati, M. Gheorghiu, and M. Chechik, "Thorough checking revisited." U of Toronto, Tech. Rep, CSRG-540, 2006.
[14] S. Kleene, *Introduction to Metamathematics*. New York: Van Nostrand, 1952.
[15] D. Kozen, "Results on the propositional μ-calculus," *TCS*, vol. 27, pp. 334–354, 1983.
[16] E. Clarke, E. Emerson, and A. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM TOPLAS*, vol. 8, no. 2, pp. 244–263, 1986.
[17] K. Larsen and B. Thomsen, "A modal process logic," in *LICS*, 1988, pp. 203–210.
[18] E. Emerson and C. S. Jutla, "Tree automata, mu-calculus and determinacy," in *FOCS*, 1991, pp. 368–377.
[19] A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. Marrero, "An improved algorithm for the evaluation of fixpoint expressions," *TCS*, vol. 178, no. 1–2, pp. 237–255, 1997.
[20] I. Walukiewicz, "Notes on the propositional μ-calculus: Completeness and related results," BRICS, NS-95-1, Tech. Rep., 1995.
[21] G. Bhat, R. Cleaveland, and A. Groce, "Efficient model checking via buchi tableau automata," in *CAV*, 2001, pp. 38–52.
[22] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.