Model Checking on Trees with Path Equivalences[*]

Rajeev Alur, Pavol Černý, and Swarat Chaudhuri

University of Pennsylvania

Abstract. For specifying and verifying branching-time requirements, a reactive system is traditionally modeled as a labeled tree, where a path in the tree encodes a possible execution of the system. We propose to enrich such tree models with "jump-edges" that capture observational indistinguishability: for an agent a, an a-labeled edge is added between two nodes if the observable behaviors of the agent a along the paths to these nodes are identical. We show that it is possible to specify information flow properties and partial information games in temporal logics interpreted on this enriched structure. We study complexity and decidability of the model checking problem for these logics. We show that it is PSPACE-complete and EXPTIME-complete respectively for fragments of CTL and μ-calculus-like logics. These fragments are expressive enough to allow specifications of information flow properties such as "agent A does not reveal x (a secret) until agent B reveals y (a password)" and of partial information games.

1 Introduction

Temporal logics have been successfully used for specifying and verifying requirements of reactive systems such as distributed protocols [6, 12]. In particular, in the branching-time approach, a system is modeled as a labeled tree whose paths correspond to executions of the system; a specification describes a set of correct trees; and verification reduces to a membership question [10]. Typical branching-time specification languages include CTL, the μ-calculus, and tree automata [9, 7]. The theoretical foundations of this approach are now well understood, and model checkers such as SMV implement highly optimized algorithms for verifying branching-time requirements of finite-state systems [3, 5].

This paper is motivated by our interest in extending model checking to reasoning about secrecy requirements of software systems [14]. Informally, a variable x is not secret after an execution e of a process a if the value of x is the same after all executions that are equivalent to e, where two executions are considered equivalent if the "observable" behavior of the process a (such as messages sent and received by a) is identical along the two executions. Classical tree logics cannot relate distinct paths in the tree, and thus, secrecy is not specifiable in logics such as the μ-calculus [1].

To be able to specify properties such as secrecy, we propose to enrich the traditional tree model with "jump-edges" that capture observational indistinguishability. More precisely, consider a tree T whose nodes are labeled with sets of atomic propositions. For an agent a, if the set of propositions $O(a)$ captures the observable behavior of a, then two tree nodes are considered a-equivalent if the paths from the root to these nodes agree on the propositions in $O(a)$ at every step. We convert the tree T into a graph $IG(T)$ by adding, for every agent a of interest, an a-labeled edge between every pair of a-equivalent nodes. One can view $IG(T)$ as a Kripke model, where both nodes and edges have labels, and interpret standard tree logics over it. For an agent a, we also define a stuttering (weak) equivalence on paths to make modeling of timing insensitive information flow properties possible. We define a graph $IG^w(T)$ similarly.

Tree logics interpreted over tree models augmented with equivalence edges have rich expressiveness. To specify that the agent a keeps the value of a variable x secret, we simply have to assert that for all tree nodes, the value of x is different from the value of x in one of the nodes connected by an a-labeled edge. One can integrate temporal reasoning with secrecy to specify requirements such as "agent a does not reveal x unless agent b reveals y." These examples, as well as the more specific examples in Section 2, lead us to conclude that a tree with path equivalences is a useful model for reasoning about information flow properties. The reason is that it contains just enough information so that these properties are specifiable in logics interpreted on this model.

Games are useful for specifying requirements as well as for formulating synthesis questions. In *partial information* games, the strategy can depend only on the sequence of observations, rather than the complete execution of the system. If a-labeled edges model the knowledge of player a (i.e. they connect two nodes in the tree iff along the paths leading to these nodes the sequence of observations of a is the same), then different versions of such partial information games are also expressible in our framework.

In our formulation, the model checking question is to decide whether $IG(T_K)$ satisfies a tree logic formula φ, where T_K is the tree unfolding of a finite-state model K. Keeping track of paths equivalent with respect to one agent requires a subset construction leading to PSPACE complexity. We show that this construction can be generalized, and the key parameter is the *nesting depth* of the specification. Informally, when we need to evaluate a formula φ after jumping across an a-labeled edge, then an additional layer of subset construction is required to process b-equivalence, for agents $b \neq a$. We show that, if we restrict the nesting depth to 1, as is the case for all our example specifications, the model checking problem for a CTL-like logic is PSPACE-complete, and EXPTIME-complete for a μ-calculus-like logic. When nesting depth is unbounded, model checking for CTL$\approx$ (the CTL-like logic) becomes nonelementary, and is undecidable for $\mu_\approx$-calculus (the μ-calculus-like logic).
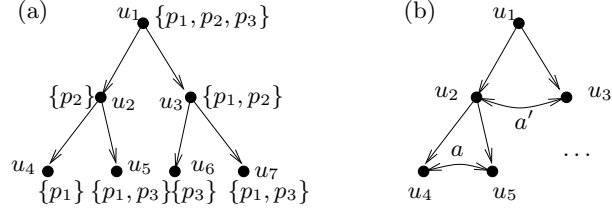
Fig. 1. (a) A labeled tree   (b) Part of its equivalence graph

2 Trees with path equivalences

Let P be a set of propositions. We consider labeled, unranked, unordered, infinite trees of the form $T = (V, E, \lambda, r)$, where V is an infinite set of nodes, $E \subseteq V \times V$ is a set of tree edges, $\lambda : V \to 2^P$ is a map labeling each node with the set of propositions holding there, and $r \in V$ is the root of the tree. A *path* in T is a sequence of nodes $\pi = v_0 v_1 v_2 \ldots$ such that $v_0 = r$ and for all i, v_i is the parent of v_{i+1}. Note that each node can be associated with a unique path (the path that leads from the root to this node) and vice-versa.

Let A be a fixed set of agents, and let us have a map $O : A \to 2^P$ defining the set of observables for an agent. We use the map O to define equivalences among paths in a tree T as follows. Let the map $Obs_a : V \to 2^P$, defined as $Obs_a(v) = \lambda(v) \cap O(a)$ for all v, return the observables of a at a node v of T. We lift this map to paths in T by defining $Obs_a(v_0 v_1 \ldots) = Obs_a(v_0) Obs_a(v_1) \ldots$. Let u and v be two nodes of T and let π be a path leading from the root to u and π' a path leading from the root to v. Nodes u and v are *a-equivalent* (written as $u \approx_a v$) iff $Obs_a(\pi) = Obs_a(\pi')$.

We define the *equivalence graph* $IG(T)$ of a tree T as the node and edge-labeled graph where: (1) the set of nodes is the set V of nodes of T; (2) the *root node* of $IG(T)$ is the root r of T; (3) the node-labeling map λ is the same as in T; (4) there is an *unlabeled edge* from node u to node v (in this case, we write $u \to v$) iff (u, v) is an edge in T; (5) for each agent a, there is an edge labeled a from u to v (we write $u \xrightarrow{a} v$) iff $u \approx_a v$. Intuitively, the structure $IG(T)$ uses a-labeled edges to capture equivalence and defined by the relation $\approx_a$. We can now view $IG(T)$ as a Kripke structure rooted at r. It is on this structure that we interpret our logics. Fig. 1-(a) depicts a tree T with path equivalences. We have two agents a and a' satisfying $O(a) = \{p_1, p_2\}$ and $O(a') = \{p_2, p_3\}$, and the nodes $u_1, u_2, \ldots$ are labeled as in the figure. Now it is easy to check that, for instance, $u_2 \approx_{a'} u_3$. Consequently, the edges of the equivalence graph $IG(T)$, part of which is shown in Fig. 1-(b), include $u_2 \xrightarrow{a'} u_3$ (and $u_3 \xrightarrow{a'} u_2$.)

The above definition of a-equivalence can be considered time sensitive in the sense that it can model an observer who knows that a transition has occurred even if the observation has not changed. We consider also the following time insensitive equivalence. Let $\equiv_w$ be the smallest congruence on sequences of sets

of propositions such that $U \equiv_w UU$, where U is a set of propositions. This relation is sometimes called stuttering congruence. Once more, let u and v be two nodes of T and let π be a path leading from the root to u and π' a path leading from the root to v. Nodes u and v are *weakly a-equivalent* (written as $u \approx_a^w v$) iff $Obs_a(\pi) \equiv_w Obs_a(\pi')$. The *weak-equivalence graph $IG^w(T)$* graph is defined similarly as $IG(T)$, with $\approx_a^w$ replacing $\approx_a$.

3 Branching-time logics on equivalence graphs

In this section, we interpret branching-time temporal logics on equivalence graphs and apply this interpretation to express some natural information-flow and partial-information requirements.

$\mu_\approx$**-calculus** The $\mu_\approx$-calculus has modalities to reason about edges labeled a, for any agent a, as well as unlabeled edges. For example, we have formulas such as $\langle a \rangle \varphi$, which holds at a node u iff there is a node v satisfying φ such that $u \xrightarrow{a} v$. In order to increase the expressiveness of the logic (without increasing the complexity of model checking), we add an operator $\langle \bar{a} \rangle$ to the syntax. The formula $\langle \bar{a} \rangle \varphi$ holds at a node u if there is another node v satisfying φ on the same level of $IG(T)$ (i.e., with the same distance from the root) that is not a-equivalent to u. See Example 4 below for an example of a property specified using the $\langle \bar{a} \rangle$ operator. To define the semantics of this operator, we will need to refer to nodes that are on the same level. This can be done using an agent sl such that $O(sl) = \emptyset$. Intuitively, this agent does not observe anything, and thus considers all the nodes at the same level to be equivalent.

Formally, let P be the set of propositions labeling our trees, and *Var* be a set of *variables*. Formulas in the $\mu_\approx$-calculus are given by the grammar: $\varphi = p \mid \neg \varphi' \mid X \mid \varphi_1 \vee \varphi_2 \mid \langle \, \rangle \varphi' \mid \langle a \rangle \varphi' \mid \langle \bar{a} \rangle \varphi' \mid \mu X. \varphi'(X)$, if X occurs in φ' only under an even number of negations, where $p \in P, a \in A$ and $X \in Var$.

As for semantics, consider the equivalence graph $IG(T)$ of a tree T with path equivalences. A formula φ is interpreted in an *environment* $\mathcal{E}$ that interprets free variables of the formula as sets of nodes in $IG(T)$. The set $[\![\varphi]\!]_\mathcal{E}$ of nodes satisfying φ in environment $\mathcal{E}$ is defined inductively in a standard way. We state only a few cases: (1) $[\![\langle \, \rangle \varphi]\!]_\mathcal{E} = \{u : \text{for some } v, u \to v \text{ and } v \in [\![\varphi]\!]_\mathcal{E}\}$; (2) $[\![\langle a \rangle \varphi]\!]_\mathcal{E} = \{u : \text{for some } v, u \xrightarrow{a} v \text{ and } v \in [\![\varphi]\!]_\mathcal{E}\}$, (3) $[\![\langle \bar{a} \rangle \varphi]\!]_\mathcal{E} = \{u : \text{for some } v, u \xrightarrow{sl} v \text{ and } \neg(u \xrightarrow{a} v) \text{ and } v \in [\![\varphi]\!]_\mathcal{E}\}$. If φ is a closed formula, its satisfaction by u is independent of the environment. If u satisfies φ in this case, then we write $u \models \varphi$. If $IG(T)$ has root r, then T satisfies φ ($T \models \varphi$) iff $r \models \varphi$.

$\mu_\approx^w$**-calculus** For reasoning on the model $IG^w(T)$, we use a fragment of $\mu_\approx$-calculus called $\mu_\approx^w$-calculus that does not contain the operator $\langle \bar{a} \rangle$, since in this case the same level predicate is not meaningful. If the root r of $IG^w(T)$ satisfies a closed $\mu_\approx^w$-calculus formula φ, then T satisfies φ (written $T \models \varphi$).

CTL$\approx$ As we shall see in Section 4, the full $\mu_\approx$-calculus over equivalence trees turns out to have an undecidable model checking problem[1]. Consequently,

---

[1] One may wonder if monadic second order logic (MSO) is of any interest in this context. It turns out that a single path equivalence relation suffices to encode the

we are interested in a simple fragment called CTL$\approx$ that is very similar to CTL interpreted on equivalence trees. Not only is this logic decidable, but it is also expressive enough for most of our illustrative examples.

Formulas of CTL$\approx$ are given by: $\varphi = p \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi' \mid EX\,\varphi' \mid EI_a\,\varphi' \mid EI_{\bar{a}}\,\varphi' \mid \varphi_1\,EU\,\varphi_2 \mid EG\,\varphi'$, where $p \in P$ and $a \in A$ as before. Following CTL conventions, let us use the following abbreviations $EF\,\varphi$ and $AG\,\varphi$. We also write $AX\,\varphi$, $AI_a\,\varphi$ and $AI_{\bar{a}}\varphi$ as shorthand for $\neg EX\,\neg\varphi$, $\neg EI_a\,\neg\varphi$, and $\neg EI_{\bar{a}}\,\neg\varphi$. We define the semantics of CTL$\approx$ using a map $\Psi : \varphi \mapsto \psi$ that rewrites a CTL$\approx$ formula φ as a $\mu_{\approx}$-calculus formula ψ. The function Ψ is defined inductively in the standard way. We state the definition only for a few cases: $\Psi(EI_a\,\varphi') = \langle a \rangle \Psi(\varphi')$ and $\Psi(EI_{\bar{a}}\,\varphi') = \langle \bar{a} \rangle \Psi(\varphi')$. A tree T with path equivalences satisfies a CTL$\approx$ formula φ iff it satisfies $\Psi(\varphi)$.

CTL$\approx^w$ We also consider the logic CTL$\approx^w$ for reasoning on the model with weak path equivalences $IG^w(T)$. This logic does not contain the operator $EI_{\bar{a}}$, but otherwise is same as CTL$\approx$. Its semantics is defined on $IG^w(T)$.

Semantics on finite Kripke structures. We use finite Kripke structures to model finite-state systems. Formally, a Kripke structure K is a tuple $(Q, \rightarrow \subseteq Q \times Q, \lambda : Q \rightarrow 2^P, r)$, where Q is a finite set of states, $\rightarrow$ is a transition function, $\lambda : Q \rightarrow 2^P$ is a map labeling each state with the set of propositions, and $r \in Q$ is the initial state.

We want to define when a Kripke structure K satisfies a CTL$\approx$ (CTL$\approx^w$,$\mu_{\approx}$,$\mu_{\approx}^w$) formula φ. Note that it is not possible to define whether or not the formula holds in a particular state of K. The reason is that the equivalence relations are relations on paths in the structure, rather than on states of the structure. Thus, given a state s, it is not possible to determine which states are equivalent to s. This also implies that whether or not a given Kripke structure K satisfies φ can be defined inductively on the structure of φ on the tree unrolling of K. For a node in this tree, there is a unique path leading to it, so the set of equivalent nodes is well-defined. Given a Kripke structure K, let T_K be its tree unrolling. T_K can be seen as a tree with (weak) path equivalences (which are determined by the set of agents A). Then we define $K \models \varphi$ iff $T_K \models \varphi$.

Let us now see how logics on trees with path equivalences aid specification.

Example 1. Consider the game of Battleship. In our formulation, each player owns a grid whose cells are filled with 0's and 1's, and at each round, a player asks another player about the contents of a cell. A central requirement is that player a does not reveal information about the contents of a cell (i, j) at any time unless the opponent asks specifically for them. To see how this property may be unintentionally violated in an automated Battleship game, consider an implementation where rows in a's grid are represented as linked lists that a iterates through to answer a query about a cell. Now, if a is asked about an element in an empty row, it gives an answer immediately (as it has nothing to iterate over). If the row is non-empty, it must iterate through a non-empty

---

"same-level" predicate on trees studied in the literature [11]. This implies that model checking MSO on trees with path equivalences is undecidable even for single-agent systems.

list and spend more time "thinking". Thus, a's opponent may glean information about whether a row in a's board is nonempty by tracking the time a takes to answer a query.

We can write a requirement forbidding the above scenario in CTL$\approx$. Let propositions c_{ij} and ask_{ij} be true at points in a play respectively iff cell c_{ij} contains 1 and a receives a request to reveal the contents of cell (i, j). We omit the full definition of a-equivalence in this version; roughly, observables of a includes the requests a receives, the answers it gives, and a "silent proposition" τ that holds when a is "thinking". Now consider the CTL$\approx$ property $\varphi = \neg(\neg ask_{ij}\, EU\, (AI_a\, c_{ij} \vee AI_a\, \neg c_{ij}))$, which asserts that there is no play with a node such that: (1) all behaviors a-equivalent to the play till this point lead to nodes where the content of (i, j) is the same, and (2) no explicit request for the contents of cell (i, j) is made by the opponent prior to this point. This ensures that the adversary cannot infer the contents of (i, j) by watching a's observables. On the contrary, in the case when $AI_a\varphi$ holds at any reachable node of the tree for some secret property φ, then an observer of a can infer the property φ by watching a's actions till that point. In other words, a *leaks* the secret φ.

Example 2. Logics on trees with path equivalences may be used to specify properties of systems where participants have partial information. Consider a Kripke structure representing a *blindfold reachability game* played by an agent a. At each round, an *active node* represents the current state of the game, and when a takes an action, a child of the current active node becomes the new active node. Because of partial information, however, a given action may cause different children of the current node to become active. We say that a has a winning strategy in this game if it can decide on a sequence of actions a priori, execute actions in it in succession, and no matter what actual path in the tree is taken, end in a node satisfying a target proposition p. Letting two paths be a-equivalent iff they agree on the sequence of actions of a, we find that a has a winning strategy in this game iff the tree satisfies the CTL$\approx$ requirement $EF\, (AI_a\, p))$.

Now consider an *adaptive reachability game*, where a can choose actions to guide the game while it is in progress. Let some of the tree nodes be now labeled with a *control proposition b*. At each round, a is now able to pick, along with an action, one of the *control formulas b* and $\neg b$. At any given point, partial information may cause different children of the current node to become active; however, the new active node is guaranteed to satisfy the control formula chosen at the current round. Let us define a-equivalence as before. It can be shown that a has a strategy to reach a node satisfying a target proposition p iff the game tree satisfies the $\mu_\approx$-calculus formula $\varphi = \mu X.(p \vee [a]\,[\,](b \wedge X) \vee [a]\,[\,](\neg b \wedge X))$.

Example 3. In various protocols involving multiple agents, a need for properties involving secrecy and time arises often. For example in the case of auction protocols (studied in security literature, see e.g. [4]), the following property is important. Agent a's bid is not revealed before the auctioneer reveals all the bids. In order to illustrate how such requirements can be expressed in our logic, we present the following formula, which states that agent a does not reveal p (a

secret) before agent b reveals q: $\varphi = \neg((EI_b\, q \wedge EI_b\, \neg q)\, EU\, (AI_a\, p \vee AI_a\, \neg p))$. The formula expresses it is not the case that: b does not reveal q ($EI_b\, q \wedge EI_b\, \neg q$) until a reveals p ($AI_a\, p \vee AI_a\, \neg p$). Now let us consider agents who make only time-insensitive observations, i.e. ones who cannot tell that an agent has performed an operation if the observables have not changed. This can be modeled using the weak-equivalence graph. The correctness of the protocol can thus be established by model checking the formula φ on $IG^w(T)$.

Example 4. Consider a system that is being observed by a low-security observer. We define low-security (low) and high-security (high) variables, where low variables are visible to the observer and high variables are not. We now show how to specify in CTL$\approx$ the following requirement R: " The sequence of valuations of the low variables is the same along all execution paths." Consider for example the case when there is a secret input, i.e. an input to a high variable. If the above requirement R is satisfied, the observer cannot infer any property of the secret input, since there cannot be any flow of information from the high input to low variables. (Note however that the requirement R is even stronger, it prevents e.g. inputs to low variables.)

The values of variables are encoded by propositions from a set P. We have one proposition for every bit of every variable. We will use only one agent a. The subset of propositions observable by the agent is the set of all those propositions that encode low variables. The requirement R is satisfied iff the following CTL$\approx$ formula holds: $AG\, AI_{\bar{a}}\, false$. This property says that for each node, there does *not* exist an a-nonequivalent node at the same level of the execution tree. This implies that all nodes at the same level are a-equivalent, and therefore have the same valuations of low variables. Notice that this property cannot be captured without the $AI_{\bar{a}}$ operator, since we need to refer to all nodes at the same level.

4 Model checking

In this section we present a model checking algorithm for CTL$\approx$ and the $\mu_{\approx}$-calculus. We are given a finite state system, such as a program or a protocol and a CTL$\approx$ formula φ. We want to check whether the system satisfies the formula.

Recall that $K \models \varphi$ is defined in terms of an infinite state structure. However, we can still apply model checking on a finite state system. This is because for a given CTL$\approx$ formula φ and a given Kripke structure K, we can find a finite model $FM^\varphi(K)$ such that $FM^\varphi(K) \models \varphi$ iff $T_K \models \varphi$. Let A_φ be the set of agents that appear in φ.

The *nesting depth* of a CTL$\approx$ formula φ is intuitively the number of nestings between equivalence operators EI_a, $EI_{\bar{a}}$. The only exception is the nesting of EI_a operators for the same agent, which does not contribute to nesting depth. For example, the nesting depth of $EI_a\, p$ is 1, $(EI_a\, p)\, EU\, (EI_b\, r)$ is also 1, while for $EI_a\, EI_{\bar{a}}\, p$ it is 2. On the other hand, $EI_a\, EI_a\, p$ and $EI_a\, (\varphi_1\, EU\, EI_a\, \varphi_2)$ have a nesting depth of 1. The nesting depth of φ will be denoted by $nd(\varphi)$. Formally, the nesting depth is defined as follows. We will use an auxiliary function that

takes two parameters: $nd(\varphi, a)$, where a is an agent. Let c be an agent that does not appear in φ. The function $nd(\varphi, a)$ is then defined as follows: (1) $nd(\varphi, a) = 0$ if $\varphi = p$, (2) $nd(\varphi, a) = nd(\varphi_1)$ if $\varphi = \neg(\varphi_1), EX\,\varphi_1, EI_a\,\varphi_1, EG\,\varphi_1$, (3) $nd(\varphi, a) = \max(nd(\varphi_1, a), nd(\varphi_2, a))$ if $\varphi = \varphi_1 \vee \varphi_2, \varphi_1\,EU\,\varphi_2$, (4) $nd(\varphi, a) = nd(\varphi_1, b) + 1$ if $\varphi = EI_b\,\varphi_1$ where $b \neq a$, (5) $nd(\varphi, a) = nd(\varphi_1, c) + 1$ if $\varphi = EI_{\bar{a}}\,\varphi_1$. $nd(\varphi)$ can then be defined as $nd(\varphi, c)$.

The complexity of model checking of a CTL$\approx$ formula φ grows rapidly with the nesting depth of φ. However, as we show, the nesting of EI_a operators for the same agent does not contribute to the growth in complexity of the problem. This distinction is especially important in the case of the $\mu_\approx$-calculus, where the formulas with unbounded nesting depth are undecidable in general. However, formulas where only $\langle a \rangle$ operators for the same agent are nested unboundedly are in a decidable (EXPTIME-complete) fragment. This fragment allows e.g. specification of adaptive partial-information games (see Section 3).

Finite model $FM^\varphi(K)$ We first give the intuition behind the construction of the finite state model $FM^\varphi(K)$. The states of this model carry enough information so that the semantics of CTL$\approx$ formulas can be defined on these states in such a way that $FM^\varphi(K) \models \varphi$ iff $T_K \models \varphi$. Consider the case when φ is a CTL formula. To determine whether φ holds at a node s of T_K, we only need to know to which state of K the node s corresponds, because if two nodes in T_K correspond to the same state of K, they satisfy the same CTL formulas. Now consider $\varphi \equiv EI_a\,\varphi_1$, where φ_1 is a CTL formula. Let S be the set of a-equivalent nodes of T_K. In order to determine whether $EI_a\varphi_1$ holds at s, one needs to know to which state of K the node s corresponds and to which states of K the nodes in S correspond. The amount of information needed is thus finite, and can be stored as a pair (s, U) such that $s \in Q, U \subseteq Q$, where Q is the set of states of K. We also need to know how to update this information across transitions. There are two key ideas: First, the transition relation $(s, U) \to (t, V)$ on these pairs can be computed locally - the set of nodes V equivalent to t will be all those nodes v that have the same observation as t and that have predecessors equivalent to s, i.e. stored in U. Second, we can also define an a-transition $(\xrightarrow{a})$ on these tuples locally, since the tuple stores the set of nodes that are mutually a-equivalent. The transition is thus defined as follows: $(s, U) \xrightarrow{a} (t, U)$ for $t \in U$.

This construction lends itself to generalization in three ways: we can have multiple agents, we can store information needed for $\bar{a}$ transitions, and we can keep enough information to allow nesting of equivalence and nonequivalence operators. This leads to a definition of the finite-state model of $FM^\varphi(K)$. Note that in order to allow for nesting of equivalence operators, it is not enough to store only a set of a-equivalent nodes U for all agents a. In fact, for each node in U, we need to store the set of its b-equivalent nodes (where $b \neq a$), etc. We store this information as a tree whose nodes are labeled by states of K. Formally, we define $FM^\varphi(K)$ as follows:

States of $FM^\varphi(K)$: A state W of $FM^\varphi(K)$ is a tree of depth at most $nd(\varphi)$. The vertices of these trees are labeled by states of K and edges are labeled by a or $\bar{a}$, where a is in A_φ. We require that if a subtree is an a-child of its parent,

then it itself does not have a-children. For all nodes in W, we require that no two of its a-children are isomorphic (similarly for $\bar{a}$-children). The state W is labeled by the same propositions as its root in the original Kripke structure K.

The intuition behind the definition is simple: a node s in T_K corresponds to a state W, if s is a root of W, the a-equivalent nodes of s correspond to a-children of W and this correspondence continues to depth $nd(\varphi)$. Such a state thus carries enough information to allow checking whether or not φ of nesting depth $nd(\varphi)$ holds. An example of a state W is in Figure 2. It stores the information about a node s, which has two a-equivalent nodes u and v, one a-nonequivalent node t and one b-equivalent node x.

If a subtree rooted at u is an a-child of its parent s, it does not need to have a-children, since the nodes that are a-equivalent to u are a-equivalent to s., thus we do not need to replicate these nodes as children of u. In fact, we do not replicate this information. The main reason is that for a subtree of depth d, the a-siblings store more information (they are trees of depths (at most) d) than would a-children - subtrees of depth (at most) $d-1$. This is what allows arbitrary nesting of EI_a (or $\langle a \rangle$) operators for the same agent a.

We can bound the number of states in $FM^\varphi(K)$. To state an upper bound, we will use the following function exp: $\exp(a,b,0) = a$, $\exp(a,b,n+1) = a * b * 2^{\exp(a,b,n)}$. Considering how a state is constructed (it does not have isomorphic a-children), we can conclude that $FM^\varphi(K)$ has less than $\exp(|K|, 2*|A_\varphi|, nd(\varphi))$ states.

Transition relation of $FM^\varphi(K)$: We explained above how a transition function is determined locally for tuples of the form (s, U) representing the node and a set of its a-equivalent nodes. The construction can be generalized to states of $FM^\varphi(K)$. We abuse the notation slightly and use the same notation for transition relations $\rightarrow, \xrightarrow{a}, \xrightarrow{\bar{a}}$ as is used in T_K. Given a state W, $root(W)$ refers to its root (a node in K). a-child of W refers to the tree rooted at a node that is an a-child of the root of W. For a state W of depth n, transition relation $\rightarrow$ is defined recursively on n.

- $n = 0$: Trees W and W' are of depth 0, i.e. they contain only a root without any children. $W \rightarrow W'$ if $root(W) \rightarrow root(W')$ in the Kripke structure K.
- $n = k + 1$: $W \rightarrow W'$ iff $root(W) \rightarrow root(W')$ in K and
  - V is an a-child of W' iff $Obs_a(root(W')) = Obs_a(root(V))$ and there exists an a-child U of W, such that $U \rightarrow V$
  - V is an $\bar{a}$-child of W' iff either there exists an $\bar{a}$-child U of W, such that $U \rightarrow V$ or there exists an a-child U', such that $U' \rightarrow V$ and $Obs_a(root(W')) \neq Obs_a(root(V))$.

An example of a transition $W \rightarrow W'$ transition in $FM^\varphi(K)$ is in Figure 2. The figure captures the following situation: There is a transition in K from s (the root of W) to s', and from the a-equivalent node u to u' and the node u' is a-equivalent to s' (similarly for the b-equivalent node x). The node v is a-equivalent to s and it has a transition to v' in K. However, v' is not a-equivalent to s'. The node t is non-equivalent to s, thus its successor t' will be nonequivalent to s'. The subtrees T_u, T_v, T_t, T_x need to be transformed in a similar way.
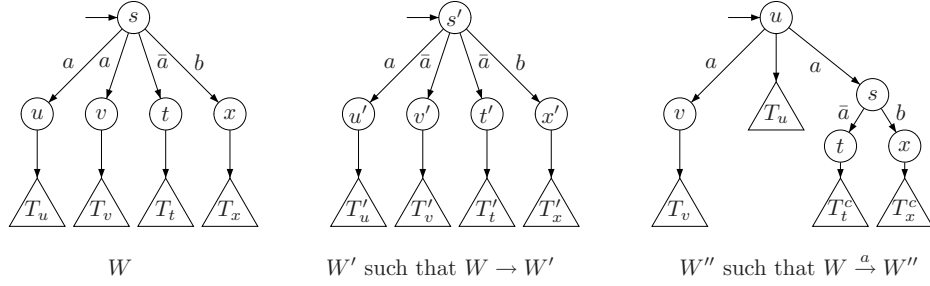
Fig. 2. States and transitions of $FM^\varphi(K)$

We defined the structure $FM^\varphi(K)$ in order to keep information about a-equivalent nodes locally. Now we use this information to define a-transitions (transitions of the form $W \xrightarrow{a} W'$). The idea is that on an a-transition, we go from a state W to a state W' represented by an a-child of W. In general, this transition leads from a tree of depth n to a tree with depth $n-1$ (this is true for b-children where $b \neq a$ and all $(\bar{a})$-children). However, for a-children we leverage the fact that a-children of a parent are mutually equivalent, which enables us to construct a tree such that the depth of a-children does not decrease. Transition relations $\xrightarrow{a}, \xrightarrow{\bar{a}}$ are defined as follows:

- $W \xrightarrow{a} W'$ iff W' can be constructed as follows: Let V be an a-child of W. Let V' be V with other a-children of W as a-children (note that V did not have a-children). Let V'' be W without all the a-children, and we remove the leaves for all the other children (to ensure that the depth of W' is smaller or equal to $nd(\varphi)$). Finally, let W' be V' with V'' as an a-child.
- $W \xrightarrow{\bar{a}} W'$ if W' is a $\bar{a}$-child of W

An example of an $\xrightarrow{a}$ transition in $FM^\varphi(K)$ $W \xrightarrow{a} W''$ is in Figure 2. The idea is that W'' will be a subtree rooted at an a-child of s, which in this case is the subtree rooted at u. However, as explained above, we add as a-children subtrees rooted at a-siblings of u (in this case, the subtree rooted at v) and the subtree rooted at the parent s and its subtrees (except the a-children). We modify these subtrees (T_t^c and T_x^c in the figure) by removing the leaves.

CTL$\approx$ We want to prove that the finite state model $FM^\varphi(K)$ is adequate for evaluating the formula φ, i.e. that for each node s of T_K there is a corresponding state W in $FM^\varphi(K)$, such that φ holds in s iff it holds in W. In order to state this claim, we need to define the correspondence between the states of T_K and $FM^\varphi(K)$. We will do so using a family of functions Ω^n. Each Ω^n is a function that relates a node in T_K to a node of $FM^\varphi(K)$. It is defined recursively as follows:

- $n = 0$: $\Omega^0(u)$ is a tree of depth 0, whose root is u.
- $n = k + 1$: $\Omega^{k+1}(u) = W$ iff W can be constructed as follows: $root(W) = u$. Consider the set S of all a-equivalent nodes of u. For every node v in this set,

compute $V = \Omega^k(v)$. Let V' be V without a-children. Add V' as an a-child to W. For every node r at the same depth as u, that is not a-equivalent to u, add $R = \Omega^k(r)$ as an $\bar{a}$-child to W.

The following lemma asserts that the construction of $FM^\varphi(K)$ is correct. It implies that model checking of CTL$\approx$ formula φ can be performed on $FM^\varphi(K)$. It is proven by induction on the nesting depth of the formula. However, the inductive hypothesis needs to be strengthened to account for the fact that arbitrary nesting of the EI_a operator for the same agent is allowed.

Lemma 1. $T_K, s \models \varphi$ iff $FM^\varphi(K), \Omega^n(s) \models \varphi$, where $n = nd(\varphi)$

A nesting-free formula is a formula with nesting depth at most 1. Thus it is a formula that can refer to operators EI_a, $EI_{\bar{a}}$ for different agents, but it can nest only the EI_a operators for the same agent. All of the example properties mentioned in Section 3 are expressed in this fragment.

Theorem 1. *The model checking problem for nesting-free formulas of CTL$\approx$ is PSPACE-complete.*

Proof. (Sketch) We show that the problem is in PSPACE using Lemma 1. The lemma shows that it is possible to reduce CTL$\approx$ model checking to CTL model checking on an exponentially larger structure $FM^\varphi(K)$, whose number of states is less than $|K| * 2 * |A| * 2^{|K|}$. Note however, that it is not necessary to construct the structure ahead of time, since the transition function can be computed locally. Thus the non-deterministic model checking algorithm for CTL [10] that uses only logarithmic space in terms of the size of the structure can be used. Therefore the model checking problem for nesting-free formulas is in PSPACE. The lower bound is obtained by reduction from equivalence checking of nondeterministic finite automata. □

The reasoning that shows that the model checking for nesting-free formulas is in PSPACE can be extended to obtain the following result.

Theorem 2. *For a fixed CTL$\approx$ formula φ such that $nd(\varphi) \geq 2$, the model checking problem is decidable in space polynomial in $exp(|K|, 2 * |A|, nd(\varphi) - 1)$.*

In order to obtain a lower bound for the model checking problem for CTL$\approx$ formulas, we can encode Shilov and Garanina's Act-CTL-K_n [15] in CTL$\approx$ and use the fact that model checking for Act-CTL-K_n has a nonelementary lower bound. Act-CTL-K_n is a logic similar to CTL with actions augmented with knowledge operators that are given the perfect-recall semantics, i.e. an agent remembers the whole sequence of its past states. CTL with actions can be encoded into CTL in a standard way. If we define the equivalence relation $\approx_a$ to be such that two paths (sequences of the multiagent system) are equivalent iff the corresponding sequences of states of agent a are the same, then the knowledge operator K_i corresponding to agent a can be encoded as follows: $K_i\varphi = \neg \ EI_a \ \neg \ \varphi$. Therefore, defining the function $Tower(n, k)$ as $Tower(n, 1) = n$ and $Tower(n, k + 1) = 2^{Tower(n,k)}$, we have:

Theorem 3. *For every algorithm $\mathcal{A}$ for the model checking problem of $CTL\approx$, and each $i \geq 1$, there is a Kripke structure K with n states and a $CTL\approx$ formula φ such that $\mathcal{A}$ runs on K and φ in time $\Omega(Tower(n,i))$.*

$\mu_\approx$**-calculus** We now consider the model checking problem for $\mu_\approx$-calculus formulas on trees with path equivalences. In general, this problem is undecidable. We can prove it by encoding Shilov and Garanina's μ-calculus of knowledge - (μPLK_n)[15]. This logic can be encoded in $\mu_\approx$-calculus over trees with path equivalences in a similar way as as Act-CTL-K_n was encoded to $CTL\approx$. Since the model checking problem for μPLK_n is undecidable, we have:

Theorem 4. *The model checking problem for the $\mu_\approx$-calculus is undecidable.*

However, we identify a decidable fragment of the $\mu_\approx$-calculus as follows. Define the set $Subf(\varphi)$ of subformulas of a formula φ inductively as: (1) for $\varphi = p$ or $\neg p$, $Subf(\varphi) = \{\varphi\}$; (2) if φ equals $\varphi_1 \vee \varphi_2$ or $\varphi_1 \wedge \varphi_2$, then $Subf(\varphi) = \{\varphi\} \cup Subf(\varphi_1) \cup Subf(\varphi_2)$; (3) if φ equals $\langle\rangle\varphi', [\,]\varphi', \langle a \rangle\varphi'$ or $[a]\varphi'$, for arbitrary a, we have $Subf(\varphi) = Subf(\varphi')$; and (4) for $\varphi = \mu X.\varphi'$ or $\nu X.\varphi'$, we have $Subf(\varphi) = \{X\} \cup Subf(\varphi')$. Now, let us only consider "well-named" formulas, i.e., closed formulas φ where for each variable X appearing in φ, there is a unique binding formula $\mu X.\varphi'$ or $\nu X.\varphi'$ in $Subf(\varphi)$ such that $X \in Subf(\varphi')$. As for the μ-calculus, every closed $\mu_\approx$-calculus formula can be rewritten in a well-named form. Now construct the graph G_φ with node set $Subf(\varphi)$ and edges as below:

1. for each node φ of the form $\langle\rangle\varphi', [\,]\varphi', \langle a \rangle\varphi', [a]\varphi', \mu X.\varphi'$, or $\nu X.\varphi'$, add an edge from φ to φ'.
2. for each node X, where $X \in Var$, add an edge from X to the unique subformula of the form $\mu X.\varphi'$ or $\nu X.\varphi'$ that binds it.

Intuitively, G_φ captures the operational semantics of φ. If there is a path from φ' to φ'' in $G_{\varphi'}$, then evaluation of φ' requires the evaluation of φ'' (note that to evaluate $\varphi' = X$, we must recursively evaluate the formula φ'' binding X).

A formula is said to be a-modal (resp. $\bar{a}$-modal) if it is of the form $\langle a \rangle\varphi$ or $[a]\varphi$ (resp. $\langle\bar{a}\rangle\varphi$ or $[\bar{a}]\varphi$). Let $\pi = \psi_1\psi_2\dots\psi_m$ be a path in G_φ. The *nesting distance* of π is k, where k is the length of the maximum subsequence $\pi' = \psi'_1\psi'_2\dots\psi'_k$ in π such that: (1) each ψ'_i is an a-modal formula for some agent a; and (2) for each i, if ψ'_i is a-modal and ψ'_{i+1} is a'-modal, then $a \neq a'$. A formula φ has *nesting depth* k if k is the least upper bound on the nesting distance of any path in G_φ. Note that such a k may not exist—if it does, then φ is said to have a *bounded* nesting depth. For instance, the formula $\varphi_1 = \nu X.([a_1]\langle a_2\rangle p \wedge \langle a_1\rangle[\,][a_1]X)$ is bounded, while $\varphi_2 = \mu X.(p \vee \langle\rangle[a_1]\langle a_2\rangle X)$ is not.

Using an argument similar to that for $CTL\approx$ and using the same structure $FM^\varphi(K)$ for formulas with nesting depth k, we can obtain a non-elementary model checking procedure for the fragment of the $\mu_\approx$-calculus with bounded nesting depth. In addition, this fragment can easily encode $CTL\approx$, so that it is non-elementary-hard. Then:

Theorem 5. *The model checking problem for a Kripke structure K and a $\mu_\approx$-calculus formula φ with nesting depth k is solvable in time $exp(|K|, 2 * |A|, k)$.*

Also, for every algorithm $\mathcal{A}$ for this problem and every $i \geq 1$, there is a Kripke structure K with n states and a formula φ such that $\mathcal{A}$ runs on K and φ in time $\Omega(\mathit{Tower}(n, i))$.

Now consider the model checking problem for *nesting-free formulas*, i.e., formulas with nesting depth 1. Recall that such formulas can express all properties involving a single agent. Now, given a Kripke structure K and a set of agents A, construct the structure $FM^\varphi(K)$ in exponential time; we can interpret nesting-free $\mu_\approx$-calculus formulas on $FM^\varphi(K)$ using the semantics of the classical μ-calculus. As above, we can show that K satisfies a nesting-free formula φ iff $FM^\varphi(K)$ satisfies φ.

For a lower bound, we turn to the model of *space-bounded private alternating Turing machines* introduced by Reif [13]. Let PALOGSPACE be the class of languages recognized by such machines using logarithmic space—Reif shows that PALOGSPACE = EXPTIME. Now recall that the alternation-free modal μ-calculus is complete for PTIME and consequently alternating LOGSPACE [10]. Augmenting this result, and encoding private tapes using the path equivalence relation induced by a single agent, we can reduce recognition by a PALOGSPACE-machine to the model checking problem for an alternation-free, single-agent $\mu_\approx$-calculus formula. The latter problem is thus EXPTIME-hard.

Theorem 6. *Model checking nesting-free $\mu_\approx$-calculus formulas is EXPTIME-complete. Model checking single-agent, alternation-free $\mu_\approx$-calculus formulas is EXPTIME-hard.*

Weak-equivalence graphs We now turn our attention to the model-checking problem on weak-equivalence graphs. The solution proceeds via the construction of $FM_w^\varphi(K)$, a finite state Kripke structure analogical to $FM^\varphi(K)$. The model checking algorithms are again based on state space exploration on the finite state model $FM_w^\varphi(K)$. For the model checking problem for CTL$\approx^w$ formulas, as well as for $\mu_\approx^w$ formulas, the same upper and lower bounds are obtained as those above for CTL$\approx$ and $\mu_\approx$ formulas.

5 Related Work

In some aspects, the logics we introduced are related to logic of knowledge [8]. The main semantic difference is that logics of knowledge are concerned about what an agent knows, whereas in the logics presented in this paper we are concerned about what an agent has revealed. However, from an intuitive point of view, it might be possible to capture what an agent a reveals by adding one "observer agent", who would observe a and record its observations (e.g. outputs and inputs of a) and then ask about the knowledge of this observer agent. However, in a finite state setting under the standard semantics for knowledge operators (the semantics is defined in terms of equivalence relations on states of the Kripke structure, not the paths), this is not possible.

The idea of introducing an observer agent would work in the case of *perfect recall* semantics [8], i.e. when an agent remembers the sequence of its past states.

In this case, our equivalence operator $EI_a\varphi$ can be translated as $\neg K_{Obs_a}\neg\varphi$, where Obs_a is the agent introduced to record the observable actions of a. Note however, that the nonequivalence operator $EI_{\bar{a}}\varphi$ cannot be expressed in logic of knowledge with perfect recall, because this logic can express properties that some or all equivalent nodes have and there is no way to refer to nonequivalent nodes. In the setting of perfect recall semantics, van der Meyden and Shilov [17] have considered model checking of LTL with knowledge operators and Shilov and Garanina [15] consider model checking of CTL and μ-calculus with knowledge operators. The construction of our finite-state model is similar to "k-trees" used in these papers. However, note that the notions of nesting depths are different, and that our notion yields better complexity bounds. (In [16], a notion of nesting depth similar to ours is used, in a context without temporal operators). We argue that our logics are more suitable for specifying secrecy and information flow properties than logics of knowledge. First, we showed that it is possible to specify information flow properties using standard tree logics (CTL, μ-calculus), provided that we enrich the tree model with path equivalences. This approach can be readily extended to other tree logics, such as ATL [2]. Second, we are also able to model information flow properties directly, without the need to introduce an observer agent for each agent in the original system. Third, some information flow properties can be expressed naturally using the $EI_{\bar{a}}$ operator. This is not possible in logic of knowledge.

For $\mu_\approx$-calculus, we have identified an EXPTIME-complete fragment in which it is possible to specify partial-information adaptive games. For simplicity, we presented our approach using Kripke structures as a basic model. However, there are other models, such as alternating transition systems (see [2]), which are better suited for modeling games. We believe our results can be easily lifted to ATSs. Note that partial information games have also been studied in the context of ATL, but were proven undecidable for multiple players.

6 Conclusion

We have introduced a branching-time logics on trees with path equivalences. We have shown that extending the execution tree by adding equivalence (or "jump") edges allows us to specify partial information games and information flow properties in tree logics (the $\mu_\approx$-calculus and CTL$\approx$). We have presented a model checking algorithm for these logics, and identified fragments where the problem has reasonable complexity (PSPACE for the case of the nesting-free fragment of CTL$\approx$).

The work presented in this paper can be extended in several directions. We plan to investigate the extension of the logics on trees with path equivalences with boolean edge formulas (a generalization of $\langle a\rangle$ and $\langle\bar{a}\rangle$ operators). Given results presented in this paper, we expect that only a (small) fragment of this generalization will be decidable. However, there are tractable fragments not explored in this paper in which one can express other information flow properties, such as noninterference and its generalizations. Another interesting direction is

to investigate automata on trees with path equivalences. We would also like to find an efficient way of implementing the model checking algorithm presented in Section 4, such as (SAT-based) bounded model checking. We plan to identify classes of applications where this implementation would prove useful and where the specifications involving multiple agents, information flow, and time are needed. Good candidates include cryptographic protocols and auction protocols. Furthermore, we would like to investigate the possibilities of extending this work for verifying information-flow properties for infinite-state systems, e.g. via abstractions that preserve information-flow properties.

References

1. R. Alur, P. Černý, and S. Zdancewic. Preserving secrecy under refinement. In *Proc. of ICALP '06*, pages 107–118, 2006.
2. R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):1–42, 2002.
3. J. Burch, E. Clarke, D. Dill, L. Hwang, and K. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
4. S. Chong and A. Myers. Decentralized robustness. In *Proc. of CSFW'02*, pages 242–256, 2006.
5. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. of CAV'02*, pages 359–364, 2002.
6. E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, pages 52–71, 1981.
7. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997.
8. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT Press, Cambridge, MA, USA, 1995.
9. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
10. O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.
11. H. Lauchli and Ch. Savioz. Monadic second order definable relations on the binary tree. *J. Symb. Log.*, 52(1):219–226, 1987.
12. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–77, 1977.
13. J. Reif. Universal games of incomplete information. In *Proc. of STOC '79*, pages 288–308, 1979.
14. F. Schneider, editor. *Trust in Cyberspace*. National Academy Press, 1999.
15. N. Shilov and N. Garanina. Model checking knowledge and fixpoints. In *Proc. of FICS'02*, pages 25–39, 2002.
16. R. van der Meyden. Common knowledge and update in finite environments. *Information and Computation*, 140(2):115–157, 1998.
17. R. van der Meyden and N. Shilov. Model checking knowledge and time in systems with perfect recall. In *Proc. of FSTTCS'99*, pages 432–445, 1999.