Distributed Synthesis for Alternating-Time Logics[⋆]

Sven Schewe and Bernd Finkbeiner

Universität des Saarlandes, 66123 Saarbrücken, Germany

Abstract. We generalize the distributed synthesis problem to the setting of alternating-time temporal logics. Alternating-time logics specify the game-like interaction between processes in a distributed system, which may cooperate on some objectives and compete on others. Our synthesis algorithm works for hierarchical architectures (in any two processes there is one that can see all inputs of the other process) and specifications in the temporal logics ATL, ATL*, and the alternating-time μ-calculus. Given an architecture and a specification, the algorithm constructs a distributed system that is guaranteed to satisfy the specification. We show that the synthesis problem for non-hierarchical architectures is undecidable, even for CTL specifications. Our algorithm is therefore a comprehensive solution for the entire range of specification languages from CTL to the alternating-time μ-calculus.

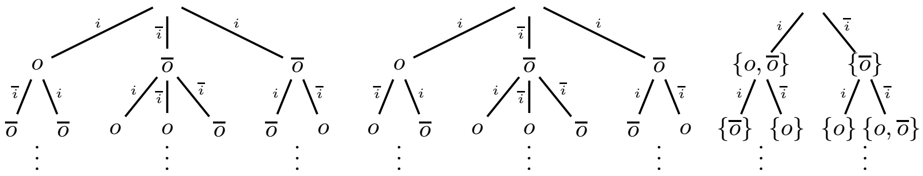
1 Introduction

Program synthesis, which automatically transforms a specification into a correct implementation, has been an active field of research since *Church's solvability problem* [1] in the early sixties. For a given sequential specification over two sets I, O of boolean input and output variables, Church's problem is to find an implementation $f : (2^I)^\omega \to (2^O)^\omega$ such that $(i, f(i))$ satisfies the specification for all possible input sequences $i \in (2^I)^\omega$. Church's problem has been intensively studied in the setting of temporal logics [2–6].

More recently, Church's problem has been extended to distributed systems [7–9], where the implementation consists of several independent processes which must choose their actions based on generally incomplete information about the system state. In game-theoretic terms, this type of synthesis solves a multiplayer game, where all players belong to the same team (when synthesizing closed systems), or where the system processes belong to one team and the external environment belongs to the other team (when synthesizing open systems).

However, in many distributed systems the processes do not consistently belong to one team, but rather form different coalitions for different objectives. In security protocols [10–12], for example, process Alice may have to deal not

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


i $\overline{i}$ i
o $\overline{o}$ $\overline{o}$
$\overline{i}$ i i $\overline{i}$ $\overline{i}$ i $\overline{i}$
$\overline{o}$ $\overline{o}$ o o $\overline{o}$ $\overline{o}$ o
i $\overline{i}$ i
o $\overline{o}$ $\overline{o}$
$\overline{i}$ i i $\overline{i}$ $\overline{i}$ i $\overline{i}$
o $\overline{o}$ o o $\overline{o}$ $\overline{o}$ o
i $\overline{i}$
$\{o, \overline{o}\}$ $\{\overline{o}\}$
i $\overline{i}$ i $\overline{i}$
$\{\overline{o}\}$ $\{o\}$ $\{o\}$ $\{o, \overline{o}\}$

(a) inconsistent strategy (b) consistent strategy (c) choice-set tree

Fig. 1. Figure 1a shows an *inconsistent* nondeterministic strategy: on the leftmost branch (with input $i \cdot \overline{i}$) the process always reacts with output $\overline{o}$, while on the rightmost branch, with identical input $i \cdot \overline{i}$, it reacts with output o. Figure 1b shows a *consistent* nondeterministic strategy. Figure 1c shows the choice-set representation of this strategy.

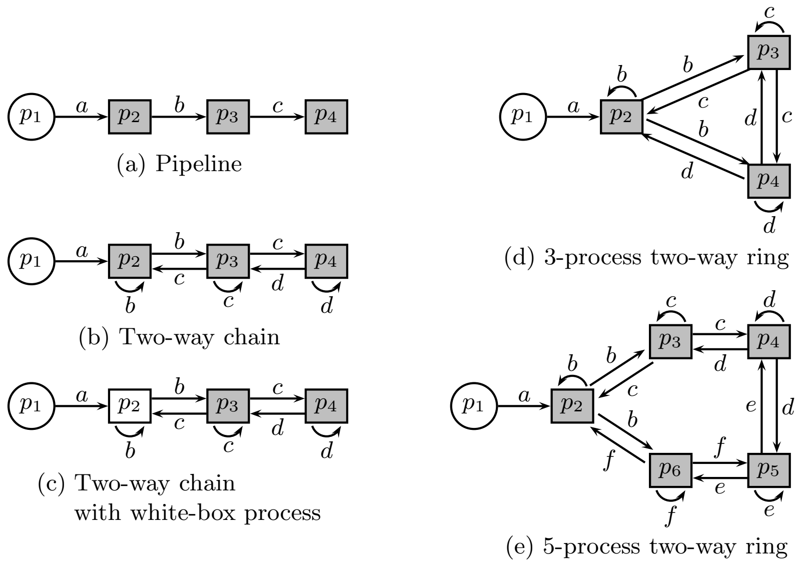
only with a hostile environment (which drops her messages from the network), but also with the dishonest process Bob, who cooperates with Alice on some objectives (like transferring money from Alice to Bob), but not on others (like delivering merchandise from Bob to Alice). Such systems can be specified with alternating-time logics, like the alternating-time μ-calculus (AMC) [13], which contain modalities expressing that a process or a coalition of processes has a strategy to accomplish a goal.

In this paper, we solve the synthesis problem for alternating-time logics. For this purpose, we generalize Church's notion of an implementation as a *deterministic strategy* or *function* $f : (2^I)^\omega \to (2^O)^\omega$ to *nondeterministic strategies* or *relations* $r \subseteq (2^I)^\omega \times (2^O)^\omega$, which allow for multiple possible outcomes due to *choices* made by the process.

Church's representation facilitates the development of automata-theoretic synthesis algorithms, because deterministic strategies can be represented as trees that branch according to the possible inputs. Each node carries a label that indicates the output of the process after seeing the input defined by the path to the node. Sets of such trees can be represented as tree automata, and can therefore be manipulated by standard tree automata operations.

Along the same lines, nondeterministic strategies can be unterstood as trees that branch not only according to inputs but also to the choices of the process. However, in this representation, sets of implementations can no longer be represented by tree automata, because tree automata cannot ensure that the choices available to the process are consistent with its observations: a strategy tree is *consistent* if every pair of nodes that are reached on paths labeled by the same input allows the *same set of choices* (for each input). For example, Figure 1a shows an inconsistent strategy tree, while the strategy tree in Figure 1b is consistent. Unfortunately, the consistent trees do not form a regular language, and can therefore not in general be recognized by tree automata.

We solve this problem with a new encoding of nondeterministic strategies as trees where *each node is labeled by the set of possible choices.* Figure 1c shows the representation of the consistent strategy of Figure 1b as such a choice-set tree. Choice-set trees always represent consistent strategies and every consistent



(a) Pipeline

(b) Two-way chain

(c) Two-way chain
with white-box process

(d) 3-process two-way ring

(e) 5-process two-way ring

Fig. 2. Distributed architectures

strategy can be represented as a choice-set tree (modulo bisimilarity). Using the choice-set representation, we define an automata-theoretic synthesis algorithm which solves the distributed synthesis problem for all *hierarchical architectures*. Let the system architecture be given as a directed graph, where the nodes represent processes, including the environment as a special process. The edges of the graph are labeled by system variables, which represent the communication of choices: the source process chooses the value and the target process is informed about the choice. The same variable may occur on multiple outgoing edges of a single node, allowing for the broadcast of information. Among the set of system processes, we distinguish two types: a process is black-box if its implementation is unknown and needs to be discovered by the synthesis algorithm. A process is white-box if the implementation is already known and fixed. Figure 2 shows several example architectures, depicting the environment as a circle, black-box processes as filled rectangles, and white-box processes as empty rectangles. We call an architecture *hierarchical* if in each pair of black-box processes, there is one process whose set of input variables is a *subset* of the set of input variables of the other process.

We show that the distributed synthesis problem for alternating-time logics is decidable if and only if the architecture is hierarchical. This is in contrast to our recent result that the distributed synthesis problem for linear and branching-time logics is decidable if and only if the informedness relation of the processes is *fork-free*, i.e., the processes can be completely ordered with respect to their relative informedness [9]. The class of architectures for which the distributed synthesis problem is decidable for alternating-time logics thus forms a *strict* *subset* of the class of architectures for which the problem is decidable for linear and branching-time logics.

For example, the pipeline architecture [7] of Figure 2a is fork-free but not hierarchical: each of the three system processes p_2, p_3, and p_4 has a unique input variable. The two-way chain [8] of Figure 2b is also fork-free and not hierarchical (process p_2 has input $\{a, b, c\}$ and process p_3 has input $\{b, c, d\}$), but becomes hierarchical if process p_2 is made white-box (process p_4 has input $\{c, d\}$, which is contained in the input of p_3), as shown in Figure 2c. Figure 2d and Figure 2e show two ring architectures: The 3-process ring of Figure 2d is both fork-free and hierarchical, while the 5-process ring of Figure 2e satisfies neither criterion.

Related Work. Synthesis algorithms for linear and branching-time logics exploit the finite-model property of these logics: a formula φ is satisfiable iff it has a finite model [3, 2]. Our construction builds on the recent result that the finite-model property extends to alternating-time logics [14, 15].

The first results for the synthesis of *distributed* systems from temporal formulas are due to Pnueli and Rosner: in their landmark paper [7] they provide a synthesis algorithm for LTL in pipeline architectures and demonstrate the existence of undecidable architectures. An automata-based synthesis algorithm for pipeline and ring architectures and CTL* specifications is due to Kupferman and Vardi [8]. We recently generalized the automata-based construction to all architectures without information forks [9].

2 The Synthesis Problem

In this paper, we solve the distributed synthesis problem for the alternating-time μ-calculus. Given an AMC formula φ and a system architecture, we decide if there exists a distributed implementation that satisfies φ.

2.1 Concurrent Game Structures

In a distributed system where all processes cooperate, we can assume that the behavior of every process is fixed *a priori*: in each state, the next transition follows a deterministic strategy. If we allow for non-cooperating behavior, we can no longer assume a deterministic choice. Instead, we fix the set of *possible decisions* and the effect each decision has on the state of the system. At each point in a computation, the processes choose a decision from the given set and the system continues in the successor state determined by that choice. For two sets of sets X and Y, let $X \oplus Y = \{x \cup y \mid x \in X, y \in Y\}$ denote the set consisting of the unions of their elements. A *concurrent game structure* (CGS) is a tuple $\mathcal{G} = (A, \Pi, S, s_0, l, \{\alpha_a\}_{a \in A}, \Delta, \tau)$, where

- $A = \mathbb{N}_k$ is a finite set of k different processes,
- Π is a finite set of atomic propositions,
- S is a set of states, with a designated initial state $s_0 \in S$,

- $l : S \to 2^{\Pi}$ is a labeling function that decorates each state with a subset of the atomic propositions, and
- α_a defines, for each process $a \in A$, a set of possible decisions.
- $\Delta : S \to \bigoplus_{a \in A}(2^{\alpha_a} \smallsetminus \{\emptyset\})$ maps each state $s \in S$ to a vector of possible decisions for the processes. For $\Delta : s \mapsto \bigoplus_{a \in A} D_a$, $\Delta_{A'}(s)$ denotes the projection of the set $\bigoplus_{a \in A} D_a$ of possible common decisions to the possible decisions $\bigoplus_{a \in A'} D_a$ of a subset $A' \subseteq A$ of the processes.
- Let $\mathfrak{D} = \bigcup_{s \in S} \Delta(s)$ denote the set of all vectors of possible decisions. Then $\tau : S \times \mathfrak{D} \rightharpoonup S$ is a (partial) transition function that maps a state s and a vector d of possible decisions for the processes to a successor state. The partial function τ is defined on $(s, d) \in S \times \mathfrak{D}$ iff $d \in \Delta(s)$.

Architectures. In a distributed system, it is not generally the case that every process is informed about the decisions of all other processes. The system architecture fixes a set of output variables for each process such that every decision corresponds to a certain value of the output variables. An output variable can be an input variable to another process, indicating that the value of the variable is communicated to that process. An *architecture* is a tuple $\mathbf{A} = (A, B, \Pi, \{I_a\}_{a \in A}, \{O_a\}_{a \in A})$ with

- a set A of processes, which is partitioned into a set $B \subseteq A$ of *black-box* processes, whose implementations we wish to synthesize, and a set $W = A \smallsetminus B$ of *white-box* processes, which have known and fixed implementations,
- a set Π of system variables or atomic propositions,
- a family $\{I_a\}_{a \in A}$ of sets of input variables, such that $I_a \subseteq \Pi$ denotes the variables visible to agent a, and
- a family $\{O_a\}_{a \in A}$ of non-empty sets of output variables that disintegrates the set Π of system variables.

An architecture is called *hierarchical* if the informedness relation $\preceq = \{(b, b') \in B \times B \mid I_b \subseteq I_{b'}\}$ is linear.

Implementations. An implementation defines for each position of a computation a subset of the output values as the set of possible decisions available to a process. The set of possible decisions must be consistent with the knowledge of the process: an *implementation* of a process $a \in A$ is a function $p_a : (2^{I_a})^* \to 2^{2^{O_a}} \smallsetminus \{\emptyset\} \equiv \mathcal{O}_a$, which assigns a choice-set of possible output values to each history of input values. Occasionally, we consider implementations that have access to a superset I of their input variables. We call a function $p_a : (2^I)^* \to \mathcal{O}_a$ with $I_a \subset I$ a *relaxed* implementation of a with input I.

We identify process implementations with trees. As usual, an Υ-tree is a prefix closed subset $Y \subseteq \Upsilon^*$ of finite words over a predefined set Υ of directions. For given sets Σ and Υ, a *Σ-labeled Υ-tree* is a pair $\langle Y, l \rangle$, consisting of a tree $Y \subseteq \Upsilon^*$ and a labeling function $l : Y \to \Sigma$ that maps every node of Y to a letter of Σ. If Υ and Σ are not important or clear from the context, $\langle Y, l \rangle$ is called a tree. If $Y \neq \emptyset$ is non-empty and, for each $y \in Y$, some successor $y \cdot \upsilon$ ($\upsilon \in \Upsilon$) of y is in Y, then Y and $\langle Y, l \rangle$ are called total. If $Y = \Upsilon^*$, Y and $\langle Y, l \rangle$ are called *full*.

A *distributed implementation* is a set $P = \{p_a\}_{a \in A}$ of process implementations, one for each process a in the architecture. A distributed implementation P defines the concurrent game structure $\mathcal{G}_P = (A, \Pi, S, s_0, l, \{\alpha_a\}_{a \in A}, \Delta, \tau)$ where

- $S = (2^{\Pi})^*$ is the full 2^{Π} tree, with its root $s_0 = \varepsilon$ as initial state,
- each state is labeled with its direction $l(s \cdot \sigma) = \sigma$ (with $l(s_0) = \emptyset$),
- $\alpha_a = 2^{O_a}$,
- $\Delta(s) = \bigoplus_{a \in A} p_a(s_a)$, where $s_a = I_1 I_2 I_3 \ldots I_n$ is the local view of process a on $s = V_1 V_2 V_3 \ldots V_n$ such that $I_m = V_m \cap I_a$ for all $m \leq n$,
- $\tau(s, d) = s \cdot d$.

2.2 Alternating-Time μ-Calculus

The alternating-time μ-calculus (AMC) extends the classical μ-calculus with modal operators which express that an agent or a coalition of agents has a strategy to accomplish a goal. AMC formulas are interpreted over concurrent game structures.

AMC Syntax. The classical μ-calculus contains two modalities, expressing that a property φ holds in some ($\Diamond\varphi$) or in all ($\Box\varphi$) successor states. In AMC[1], the operators are generalized to $\Box_{A'}\varphi$, expressing that a set $A' \subseteq A$ of agents can enforce that φ holds in the successor state, and $\Diamond_{A'}\varphi$, expressing that it cannot be enforced against the agents A' that φ is violated in the successor state.

Let P and B denote disjoint finite sets of atomic propositions and bound variables, respectively. Then

- *true* and *false* are AMC formulas.
- p and $\neg p$ are AMC formulas for all $p \in P$.
- x is an AMC formula for all $x \in B$.
- If φ and ψ are AMC formulas then $\varphi \wedge \psi$ and $\varphi \vee \psi$ are AMC formulas.
- If φ is an AMC formula and $A' \subseteq A$ then $\Box_{A'}\varphi$ and $\Diamond_{A'}\varphi$ are AMC formulas.
- If $x \in B$ and φ is an AMC formula where x occurs only free then $\mu x.\varphi$ and $\nu x.\varphi$ are AMC formulas.

AMC Semantics. An AMC formula φ with atomic propositions Π is interpreted over a CGS $\mathcal{G} = (A, \Pi, S, s_0, l, \{\alpha_a\}_{a \in A}, \Delta, \tau)$. $\|\varphi\|_{\mathcal{G}} \subseteq S$ denotes the set of nodes where φ holds. A CGS $\mathcal{G} = (A, \Pi, S, s_0, l, \{\alpha_a\}_{a \in A}, \Delta, \tau)$ is a *model* of a specification φ with atomic propositions Π iff $s_0 \in \|\varphi\|_{\mathcal{G}}$, and a distributed implementation P satisfies an AMC specification φ iff $\mathcal{G}_P$ is a model of φ.

[1] The original definition of alternating-time logics under incomplete information by Alur et al. [13] syntactically restricts the specifications such that the objectives of each process only refer to the atomic propositions that are visible to the process. This restriction ensures that the processes can *state* their respective strategies, while we only require that they *can cooperate* to accomplish their goals. For the specifications allowed in [13], the semantics coincide.

– Atomic propositions are interpreted as follows: $\|\mathit{true}\|_{\mathcal{G}} = S$, $\|\mathit{false}\|_{\mathcal{G}} = \emptyset$, $\|p\|_{\mathcal{G}} = \{s \in S \mid p \in l(s)\}$, and $\|\neg p\|_{\mathcal{G}} = \{s \in S \mid p \notin l(s)\}$.

– As usual, conjunction and disjunction are interpreted as intersection and union, respectively: $\|\varphi \wedge \psi\|_{\mathcal{G}} = \|\varphi\|_{\mathcal{G}} \cap \|\psi\|_{\mathcal{G}}$ and $\|\varphi \vee \psi\|_{\mathcal{G}} = \|\varphi\|_{\mathcal{G}} \cup \|\psi\|_{\mathcal{G}}$.

– A node $s \in S$ is in $\|\Box_{A'}\varphi\|_{\mathcal{G}}$ if the agents A' can make a joint decision $\upsilon \in \Delta_{A'}(s)$ such that, for all counter decisions $\upsilon' \in \Delta_{A \smallsetminus A'}(s)$, φ holds in the successor state.
$\|\Box_{A'}\varphi\|_{\mathcal{G}} = \{s \in S \mid \exists \upsilon \in \Delta_{A'}(s).\, \forall \upsilon' \in \Delta_{A \smallsetminus A'}(s).\, \tau(s,(\upsilon,\upsilon')) \in \|\varphi\|_{\mathcal{G}}\}$.

– A node $s \in S$ is in $\|\Diamond_{A'}\varphi\|_{\mathcal{G}}$ if for all joint decisions $\upsilon \in \Delta_{A \smallsetminus A'}(s)$ of the agents not in A', the agents in A' have a counter decision $\upsilon' \in \Delta_{A'}(s)$ that ensures that φ holds in the successor state.
$\|\Diamond_{A'}\varphi\|_{\mathcal{G}} = \{s \in S \mid \forall \upsilon' \in \Delta_{A \smallsetminus A'}(s).\exists \upsilon \in \Delta_{A'}(s).\tau(s,(\upsilon,\upsilon')) \in \|\varphi\|_{\mathcal{G}}\}$.
The modal operators $\Box$ and $\Diamond$ of the classical μ-calculus are equivalent to the modal operators $\Box_{\emptyset}$ and $\Diamond_A$, respectively.

– Let $\mathcal{G}_x^{S_x} = (A, \Pi \cup \{x\}, S, s_0, l_x^{S_x}, \{\alpha_a\}_{a \in A}, \Delta, \tau)$ denote, for $\mathcal{G} = (A, \Pi, S, s_0, l, \{\alpha_a\}_{a \in A}, \Delta, \tau)$ and $x \notin \Pi$, the adapted CGS with the labeling function $l_x^{S_x} : S \to 2^{\Pi \cup \{x\}}$, which is defined by
  - $l_x^{S_x}(s) \cap \Pi = l(s)$ and
  - $x \in l_x^{S_x}(s) \Leftrightarrow s \in S_x \subseteq S$.

  Since, for AMC formulas $\lambda x.\varphi$, x occurs only positive in φ, $\|\varphi\|_{\mathcal{G}_x^{S_x}}$ is monotone in S_x and the following least and greatest fixed points are well-defined: $\|\mu x.\varphi\|_{\mathcal{G}} = \bigcap\{S_x \subseteq S \mid \|\varphi\|_{\mathcal{G}_x^{S_x}} \subseteq S_x\}$, and $\|\nu x.\varphi\|_{\mathcal{G}} = \bigcup\{S_x \subseteq S \mid \|\varphi\|_{\mathcal{G}_x^{S_x}} \supseteq S_x\}$.

2.3 Realizability and Synthesis

We call an AMC formula φ *realizable* in a given architecture $\mathbf{A} = (A, B, \Pi, \{I_a\}_{a \in A}, \{O_a\}_{a \in A})$ and for a given set $P_W = \{p_w\}_{w \in W}$ of implementations for the white-box processes if there exists a set of implementations $P_B = \{p_b\}_{b \in B}$ for the black-box processes, such that the CGS defined by the distributed implementation $P = P_W \cup P_B$ satisfies φ. $\mathbf{A}$ is called *decidable* if realizability can be decided for all formulas φ and implementations P_W of the white-box processes.

In the following section, we present a *synthesis algorithm*, which determines if a specification is realizable. If the specification is realizable, the synthesis algorithm computes an implementation.

3 The Synthesis Algorithm

In this section, we present a synthesis algorithm for hierarchical architectures. The construction is based on automata over infinite trees and game structures.

3.1 Preliminaries: Automata over Infinite Objects

Automata over Infinite Trees. An *alternating parity tree automaton* is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, \alpha)$, where Σ is a finite set of labels, Q is a finite set of states, $q_0 \in Q$ is a designated initial state, δ is a transition function, and $\alpha : Q \to C \subset \mathbb{N}$ is a coloring function. The transition function $\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times \Upsilon)$ maps a state and an input letter to a positive boolean combination of states and directions.

The automaton runs on full Σ-labeled Υ-trees. A *run tree* $\langle R, r\rangle$ on a given full Σ-labeled Υ-tree $\langle \Upsilon^*, l\rangle$ is a $Q \times \Upsilon^*$-labeled tree where the root is labeled with (q_0, ε) and where, for each node n with label (q, y) and with the set $L = \{r(n \cdot \rho) | n \cdot \rho \in R\}$ of labels of its successors, the following condition holds: the set $\{(q', \upsilon) \in Q \times \Upsilon \mid (q', y \cdot \upsilon) \in L\}$ satisfies $\delta(q, l(y))$.

An infinite path fulfills the *parity condition*, if the highest color of the states appearing infinitely often on the path is even. A run tree is *accepting* if all infinite paths fulfill the parity condition. A total Σ-labeled Υ-tree is accepted if it has an accepting run tree.

The set of trees accepted by an alternating automaton $\mathcal{A}$ is called its *language* $\mathcal{L}(\mathcal{A})$. An automaton is empty if its language is empty.

The acceptance of a tree can also be viewed as the outcome of a game, where player *accept* chooses, for a pair $(q, \sigma) \in Q \times \Sigma$, a set of atoms of $\delta(q, \sigma)$, satisfying $\delta(q, \sigma)$, and player *reject* chooses one of these atoms, which is executed. The input tree is accepted iff player *accept* has a strategy enforcing a path that fulfills the parity condition. One of the players has a memoryless winning strategy, i.e., a strategy where the moves only depend on the state of the automaton, the position in the tree and, for player *reject*, on the choice of player *accept* in the same move.

In a *nondeterministic* tree automaton, the image of δ consists only of such formulae that, when rewritten in disjunctive normal form, contain exactly one element of $Q \times \{\upsilon\}$ for all $\upsilon \in \Upsilon$ in every disjunct. For nondeterministic automata, every node of a run tree corresponds to a node in the input tree. Emptiness can therefore be checked with an *emptiness game*, where player *accept* also chooses the letter of the input alphabet. A nondeterministic automaton is empty iff the emptiness game is won by *reject*.

Automata over Concurrent Game Structures. Generalizing symmetric automata [16], automata over concurrent game structures [15] contain *universal atoms* $(\Box, A')$, which refer to *all* successor states for *some* decision of the agents in A', and *existential atoms* $(\Diamond, A')$, which refer to *some* successor state for *each* decision of the agents *not* in A'.

An *automaton over concurrent games structures* (ACG) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, \alpha)$, where Σ, Q, q_0, and α are defined as for alternating parity automata in the previous paragraph. The transition function $\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times (\{\Box, \Diamond\} \times 2^A))$ now maps a state and an input letter to a positive boolean combination of two types of atoms: $(q, \Box, A')$ is a universal atom, and $(q, \Diamond, A')$ is an existential atom.

A run tree $\langle R, r\rangle$ on a given CGS $\mathcal{G} = (A, \Pi, S, s_0, l, \{\alpha_a\}_{a\in A}, \Delta, \tau)$ is a $Q \times S$-labeled tree where the root is labeled with (q_0, s_0) and where, for a node n with label (q, s) and a set $L = \{r(n \cdot \rho) \,|\, n \cdot \rho \in R\}$ of labels of its successors, the following property holds: there is a set $\mathfrak{A} \subseteq Q \times (\{\Box, \Diamond\} \times 2^A)$ of atoms satisfying $\delta(q, l(s))$ such that

- for all universal atoms $(q', \Box, A')$ in $\mathfrak{A}$, there exists a decision $\upsilon \in \Delta_{A'}(s)$ of the agents in A' such that, for all counter decisions $\upsilon' \in \Delta_{A \smallsetminus A'}(s)$, $(q', \tau(s, (\upsilon, \upsilon'))) \in L$, and
- for all existential atoms $(q', \Diamond, A')$ in $\mathfrak{A}$ and all decisions $\upsilon' \in \Delta_{A \smallsetminus A'}(s)$ of the agents not in A', there exists a counter decision $\upsilon \in \Delta_{A'}(s)$ such that $(q', \tau(s, (\upsilon, \upsilon'))) \in L$.

As before, a run tree is accepting iff all paths satisfy the parity condition, and a CGS is accepted iff there exists an accepting run tree.

3.2 Realizability in 1-Black-Box Architectures

We first consider the realizability problem for architectures with a single black-box process. Given such an architecture $\mathbf{A} = (A, \{b\}, \Pi, \{I_a\}_{a \in A}, \{O_a\}_{a \in A})$, an AMC specification φ and a set $P_W = \{p_w\}_{w \in W}$ of implementations for the white-box processes, the following algorithm constructs a nondeterministic automaton $\mathcal{E}$, which accepts an implementation p_b of the black-box process b iff the distributed implementation $P = P_W \cup \{p_b\}$ defines a concurrent game structure that is a model of φ. Realizability can then be checked by solving the emptiness game for $\mathcal{E}$. For convenience, we use $\mathcal{V} = \bigoplus_{a \in A} \mathcal{O}_a$ in the following constructions. The synthesis algorithm uses the following automata operations:

- **From specification to automata.** First, a specification φ is turned into an ACG $\mathcal{A}$ that accepts exactly the models of φ (Theorem 1).
- **From models to implementations.** We then transform $\mathcal{A}$ into an alternating tree automaton $\mathcal{B}$ that accepts a relaxed implementation with input Π iff it defines a model of φ (Lemmata 1 and 2).
- **Adjusting for white-box processes.** In a third step, we construct an alternating automaton $\mathcal{C}$ that accepts an $\mathcal{O}_b$-labeled 2^{Π}-tree iff the $\mathcal{V}$-labeled 2^{Π}-tree obtained by adding the decisions of the white-box processes is accepted by $\mathcal{B}$ (Lemma 3).
- **Incomplete information.** We then transform $\mathcal{C}$ into an alternating automaton $\mathcal{D}$ that accepts an $\mathcal{O}_b$-labeled 2^{I_b}-tree iff its suitable widening is accepted by $\mathcal{C}$ (Lemma 4). In the last step, we construct a nondeterministic tree automaton $\mathcal{E}$ with $\mathcal{L}(\mathcal{E}) = \mathcal{L}(\mathcal{D})$ (Lemma 5).

From Specifications to Automata. AMC formulas can be transformed to equivalent automata over concurrent game structures.

Theorem 1. *[15] Given an alternating-time μ-calculus specification φ with n subformulas, we can construct an ACG $\mathcal{A}$ with $O(n^2)$ states and $O(n)$ colors, which accepts exactly the models of φ.*

From Models to Implementations. The transformation of $\mathcal{A}$ into an alternating tree automaton that accepts a relaxed implementation iff it defines a model of φ consists of two steps: We first restrict for each process a the set of possible decisions to the fixed set $\mathcal{O}_a$ (Lemma 1) and then ensure that the label of each node reflects the preceeding decisions of the processes (Lemma 2).

Lemma 1. *For ACG $\mathcal{A} = (2^{\Pi}, Q, q_0, \delta, \alpha)$ and an architecture $\mathbf{A} = (A, B, \Pi, \{I_a\}_{a\in A}, \{O_a\}_{a\in A})$ we can construct an alternating automaton $\mathcal{A}' = (2^{\Pi} \times \mathcal{V}, Q, q_0, \delta', \alpha)$ that accepts a tree $\langle (2^{\Pi})^*, l \times \bigoplus_{a\in A} p_a\rangle$ iff the concurrent game structure $\mathcal{G} = (A, \Pi, S, s_0, l, \{\alpha_a\}_{a\in A}, \Delta, \tau)$ with $\Delta = \bigoplus_{a\in A} p_a$ and $\tau : (s, d) \mapsto s \cdot d$ is accepted by $\mathcal{A}$.*

Proof. Since the potential decisions of the processes are determined by the (relaxed) implementation, the universal and existential atoms can be resolved by boolean combinations of concrete directions.

We obtain $\delta'(q, (V, \bigoplus_{a\in A} \mathbf{O}_a))$ by resolving the $\forall\exists$ and $\exists\forall$ semantics of universal and existential atoms in $\delta(q, V)$ in the following way:

- Each occurrence of $(q', (A', \Box))$ in $\delta(q, V)$ is replaced by $\bigvee_{\bigoplus_{a\in A'} O'_a \in \bigoplus_{a\in A'} \mathbf{O}_a} \bigwedge_{\bigoplus_{a\in A\smallsetminus A'} O'_a \in \bigoplus_{a\in A\smallsetminus A'} \mathbf{O}_a} (q', \bigcup_{a\in A} O'_a)$.
  The outer disjunction refers to the fact that the agents in A' first choose a direction in accordance with the enabled directions in the current state. The inner conjunction refers to the counter choice made by the agents in $A \smallsetminus A'$.
- Likewise, each occurrence of $(q', (A', \Diamond))$ in $\delta(q, V)$ is replaced by $\bigwedge_{\bigoplus_{a\in A\smallsetminus A'} O'_a \in \bigoplus_{a\in A\smallsetminus A'} \mathbf{O}_a} \bigvee_{\bigoplus_{a\in A'} O'_a \in \bigoplus_{a\in A'} \mathbf{O}_a} (q', \bigcup_{a\in A} O'_a)$. □

Let $\langle \Upsilon^*, dir\rangle$ denote the Υ-labled Υ-tree with $dir(y \cdot \upsilon) = \upsilon$ for all $y \in \Upsilon^*$ and $\upsilon \in \Upsilon$, and $dir(\varepsilon) = \upsilon_0$ for some predefined $\upsilon_0 \in \Upsilon$.

Lemma 2. *[17] Given an alternating automaton $\mathcal{A}' = (\Upsilon \times \Sigma, Q, q_0, \delta, \alpha)$ over $\Upsilon \times \Sigma$-labeled Υ-trees, we can construct an alternating automaton $\mathcal{B} = (\Sigma, Q \times \Upsilon, q'_0, \delta', \alpha')$ over Σ-labeled Υ-trees such that $\mathcal{B}$ accepts a tree $\langle \Upsilon^*, l\rangle$ iff $\mathcal{A}'$ accepts $\langle \Upsilon^*, dir \times l\rangle$.* □

Adjusting for White-box Processes. In this step, we eliminate the trees that are inconsistent with the decisions of the white-box processes. These decisions are represented by the $\bigoplus_{w\in W} \mathcal{O}_w$ fraction of the label. We assume that the implementations $\{p_w\}_{w\in W}$ of the white-box processes are represented as a deterministic Moore machine with output alphabet $\bigoplus_{w\in W} \mathcal{O}_w$. We construct an automaton that simulates the behavior of this Moore machine, replacing the $\bigoplus_{w\in W} \mathcal{O}_w$ fraction of the label with the output of the Moore machine. The state space of this automaton is linear in the state space of the original automaton and in the state space of the Moore machine, while the set of colors remains unchanged.

Lemma 3. *[18] Given an alternating automaton $\mathcal{B} = (\Sigma \times \Xi, Q, q_0, \delta, \alpha)$ over $\Sigma \times \Xi$-labeled Υ-trees and a deterministic Moore machine $\mathcal{O}$ with set O of states and initial state $o_0 \in O$ that produces a Ξ-labeled Υ-tree $\langle \Upsilon^*, l\rangle$, we can construct an alternating automaton $\mathcal{C} = (\Sigma, Q \times O, (q_0, o_0), \delta', \alpha')$ over Σ-labeled Υ-trees, such that $\mathcal{C}$ accepts $\langle \Upsilon^*, l'\rangle$ iff $\mathcal{B}$ accepts $\langle \Upsilon^*, l''\rangle$ with $l'' : y \mapsto (l'(y), l(y))$.*
If $\mathcal{B}$ is a nondeterministic automaton, so is $\mathcal{C}$. □

Incomplete Information. The output of the black-box process b may only depend on the input I_b visible to b. For a set $\Xi \times \Upsilon$ of directions and a node $x \in (\Xi \times \Upsilon)^*$, $hide_\Upsilon(x)$ denotes the node in Ξ^* obtained from x by replacing (ξ, υ) by ξ in each letter of x. For a Σ-labeled Ξ-tree $\langle \Xi^*, l\rangle$ we define the Υ-widening of $\langle \Xi^*, l\rangle$, denoted by $widen_\Upsilon(\langle \Xi^*, l\rangle)$, as the Σ-labeled $\Xi \times \Upsilon$-tree $\langle (\Xi \times \Upsilon)^*, l'\rangle$ with $l'(x) = l(hide_\Upsilon(x))$.

Lemma 4. *[17] Given an alternating automaton $\mathcal{C} = (\Sigma, Q, q_0, \delta, \alpha)$ over Σ-labeled $\Xi \times \Upsilon$-trees, we can construct an alternating automaton $\mathcal{D} = (\Sigma, Q, q_0, \delta', \alpha)$ over Σ-labeled Ξ-trees, such that $\mathcal{D}$ accepts $\langle \Xi^*, l\rangle$ iff $\mathcal{C}$ accepts $widen_\Upsilon(\langle \Xi^*, l\rangle)$.* □

The resulting alternating automaton can be transformed into an equivalent nondeterministic automaton.

Lemma 5. *[9, 19] Given an alternating automaton $\mathcal{D}$ with n states and c colors, we can construct an equivalent nondeterministic automaton $\mathcal{E}$ with $n^{O(c\cdot n)}$ states and $O(c \cdot n)$ colors.* □

3.3 Realizability in Hierarchical Architectures

For a hierarchical architecture $\mathbf{A} = (A, B, \Pi, \{I_a\}_{a\in A}, \{O_a\}_{a\in A})$, the linear informedness relation $\preceq\; = \{(b, b') \in B \times B \mid I_b \subseteq I_{b'}\}$ partitions the black-box processes B into equivalence classes and defines an order on them. If $\preceq$ defines n different equivalence classes, we say that $\mathbf{A}$ has n levels of informedness. We define an ordering function $o : \mathbb{N}_n \to 2^B$, which maps each natural number $i \in \mathbb{N}_n$ to the set of i-th best informed black-box processes. For convenience, we use $\mathcal{O}_i = \bigoplus_{b\in o(\{i,\ldots,n\})} \mathcal{O}_b$ and $I_i = I_b$ for $b \in o(i)$.

The Algorithm. We start by applying the transformations discussed in the previous subsection (Theorem 1 and Lemmata 1 through 3) to construct a tree automaton $\mathcal{C}_0$ that accepts a set of relaxed implementations $P_0 = \{p_b\}_{b\in B}$ (with input Π) iff $P = P_W \cup P_0$ satisfies φ.

Then, we stepwise eliminate the processes in decreasing order of informedness. We successively construct:

- The alternating automaton $\mathcal{D}_i$ that accepts a $\mathcal{O}_i$-labeled 2^{I_i}-tree iff its widening is accepted by $\mathcal{C}_{i-1}$ (Lemma 4).
  A set $P_i = \{p^i_b \mid b \in B_i\}$ of relaxed implementations with input I_i for the processes in $B_i = o(\{i, \ldots, n\})$ is accepted by $\mathcal{D}_i$ iff there is a set $\overline{P}_i = \{\overline{p}^i_b \mid b \in \overline{B}_i\}$ of implementations for the processes in $\overline{B}_i = o(\mathbb{N}_{i-1})$, such that $P_W \cup P_i \cup \overline{P}_i$ satisfies φ.

- The nondeterministic automaton $\mathcal{E}_i$ with $\mathcal{L}(\mathcal{E}_i) = \mathcal{L}(\mathcal{D}_i)$ (Lemma 5); and
- The nondeterministic automaton $\mathcal{C}_i$ that accepts an $\mathcal{O}_{i+1}$-labeled I_i-tree iff it can be extended to an $\mathcal{O}_i$-labeled I_i-tree accepted by $\mathcal{C}_i$ (Lemma 6).

Narrowing and nondeterminization have been discussed in the previous section, and language projection is a standard operation on nondeterministic automata.

Lemma 6. *Given a nondeterministic automaton $\mathcal{E} = (\Sigma \times \Xi, Q, q_0, \delta, \alpha)$ that runs on $\Sigma \times \Xi$-labeled Υ-trees, we can construct a nondeterministic automaton $\mathcal{C} = (\Sigma, Q, q_0, \delta', \alpha)$ that accepts a Σ-labeled Υ-tree $\langle \Upsilon^*, l_\Sigma \rangle$ iff there is a $\Sigma \times \Xi$-labeled Υ-tree $\langle \Upsilon^*, l_\Sigma \times l_\Xi \rangle$ accepted by $\mathcal{E}$ with $\langle \Upsilon^*, l \rangle = proj_\Sigma(\langle \Upsilon^*, l_\Xi \rangle)$.*

Proof. $\mathcal{C}$ can be constructed by using δ' to guess the correct tree: we set $\delta' : (q, \sigma) \mapsto \bigvee_{\xi \in \Xi} \delta(q, (\sigma, \xi))$. □

We check realizability by solving the emptiness game for $\mathcal{E}_n$. This step can be extended to the synthesis of implementations $\{p_b\}_{b \in B}$ of the black-box processes.

3.4 Synthesis

The specification is realizable iff player *accept* has a winning strategy in the emptiness game of $\mathcal{E}_n$. From this strategy we obtain by projection a family of implementations $P_n = \{p_a \,|\, a \in o(n)\}$ for the least-informed processes.

In increasing order of informedness, we obtain implementations for the other processes: After computing implementations for the processes in $o(\{i{+}1, \ldots, n\})$, they are represented as Moore machines. Using Lemma 3, we then construct from $\mathcal{E}_i$ a nondeterministic automaton $\mathcal{F}_i$ that accepts those implementations $\widehat{P}_i$ for the processes in $o(i)$ for which there exists a set of implementations $\overline{P}_{i-1} = \{p_a \,|\, a \in o(\mathbb{N}_{i-1})\}$ such that $\overline{P}_{i-1} \cup \widehat{P}_i \cup P_{i+1}$ satisfies φ. $\mathcal{F}_i$ is nonempty by construction. From the winning strategy for player *accept* we obtain by projection a family of implementations $P' = \{p_a \,|\, a \in o(i)\}$ and set P_i to $P' \cup P_{i+1}$.

Theorem 2. *The distributed synthesis problem for an architecture* $\mathbf{A}$ *with n levels of informedness, a specification φ given as an AMC formula, and a family $P_W = \{p_w\}_{w \in W}$ of implementations of the white-box processes can be solved in time n-exponential in the number of subformulas of φ.*

Proof. The specification φ is realizable for an architecture $\mathbf{A}$ and a given set $\{p_w\}_{w \in W}$ of white-box strategies iff $\mathcal{E}_n$ is not empty. The construction of $\mathcal{E}_n$ involves one transformation of an alternating automaton to a nondeterministic automaton for each $i \in \mathbb{N}_n$, and therefore takes n-exponential time in the number of subformulas of φ. The size of each nondeterministic automaton $\mathcal{F}_i$ is linear in the size of $\mathcal{E}_i$ and the size of the Moore machines for the strategy of the less-informed processes. Each step along the order of informedness therefore again takes n-exponential time. □

The upper bounds for ATL, CTL* and ATL* follow from linear translations to alternation-free AMC [13], exponential translations to the μ-calculus [20], and doubly exponential translations to AMC [21, 13], respectively. μ-calculus and CTL form a syntactical subset of AMC and ATL, respectively.

Corollary 1. *The distributed synthesis problem for an architecture* **A** *with n levels of informedness and a specification φ can be performed in time n-exponential in the length of φ for specifications in CTL, ATL, or the classical μ-calculus, $(n+1)$-exponential in the length of φ for specifications in CTL*, and $(n+2)$-exponential in the length of φ for specifications in ATL*.*

A matching nonelementary lower bound (for LTL formulas and pipelines[2]) is provided in [7].

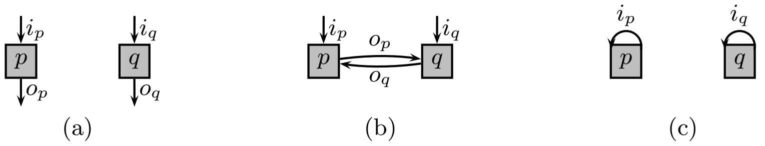
4 Completeness

In the previous section we showed that the linearity requirement on the informedness relation is a sufficient condition for the decidability of an architecture. In this section, we show that the condition is also necessary: we prove that, for non-linear informedness relations, the synthesis problem is already undecidable for the sublogic CTL.

The proof is a variant of the reduction of the synthesis problem for deterministic implementations to the halting problem in [7, 9]. In the following we give a brief sketch of this argument before discussing the extension to nondeterministic strategies. In the simplest case, shown in Figure 3a, there are two processes p and q, such that the input i_p and the output o_p of process p is invisible to process q, and, vice versa, i_q and o_q are invisible to p. For a given deterministic Turing machine M, the conjunction ψ_M of the following conditions is realizable iff M halts on the empty input tape:

- The environment can send a *start* signal through i_p and i_q.
- Initially, p and q output the terminal state of M.
- Upon receiving the first *start* signal, p (q) starts to output syntactically legal configurations of M such that
  - the first two configurations are correct, and
  - whenever p and q output two configurations C_p and C_q, such that C_p (C_q) is the successor configuration of C_q (C_p), then the next configurations emitted by p and q have the same property.
- p and q always eventually output the terminal state of M.

[2] For linear-time specifications we can restrict our attention w.l.o.g. to deterministic implementations. In this case, the processes at the beginning of the pipeline have (implicit) knowledge of the output produced by processes later in the pipeline [9]. Turning this knowledge into explicit input does not change the nonelementary complexity.



(a) (b) (c)

Fig. 3. Three undecidable situations: an architecture is undecidable if it contains two processes with incomparable sets of inputs.

For the more complicated case that the processes have access to each other's output (Figure 3b), ψ_M is extended to describe a two-phase protocol: On the input variables, a *start* symbol may be transmitted in the first phase, and an XOR key is sent in the second phase. The output variables are again used to emit sequences of configurations of Turing machines. In the first phase, the output is constantly set to true, and in the second phase it is encrypted by the last received XOR key. In this way, the processes cannot infer the decrypted meaning of the output from the other process, even if they have access to each other's output [9].

We now extend this argument to prove the undecidability of the synthesis problem for nondeterministic strategies and architectures with non-linear informedness relation. In addition to the architectures considered above, we take into account the situation where the two processes do not receive any input from an external environment (Figure 3c). In this case, we specify that the *start*-symbols and XOR keys are chosen completely nondeterministically during the first and second phase. The configurations of the Turing machine are emitted in a separate third phase, where the values of the output variables are specified to be deterministic.

Theorem 3. *The synthesis problem for CTL specifications is undecidable for all architectures with two black-box processes $b, p \in B$ with incomparable sets of input variables ($I_p \not\supseteq I_q \not\supseteq I_p$).*

Proof. The halting problem is reduced to the synthesis problem as follows. W.l.o.g. we fix one input variable for p and q that is invisible to the other process ($i_p \in I_p \smallsetminus I_q$ and $i_q \in I_q \smallsetminus I_p$) and two output variables $o_p \in O_p$ and $o_q \in O_q$. We extend the CTL specification ψ_M (from Theorem 5.3 of [9]) to describe the following three-phase communication pattern:

- A *start* symbol can be transmitted to p and q through i_p and i_q in a first phase.
- A one bit XOR key is transmitted to p and q through i_p and i_q in a second phase.
- p and q output an encoded bit of their output sequence in a third phase.

We extend the specification with the following guarantees:

- Exactly in every third round (and in the third round from the beginning) the values of the variables o_p and o_q are fixed deterministically.
  In the remaining rounds they are set nondeterministically to *true* and *false*.

- The variables in $\{i_p, i_q\} \smallsetminus \{o_p, o_q\}$ are set deterministically to *true* in every third round (and in the third round from the beginning).
  In the remaining rounds they are set nondeterministically to *true* and *false*.
- To rule out the influence of the remaining variables, we require that they are always set to *true*.

If the white-box strategies are chosen consistently with the specification, the synthesis problem has a solution iff M halts on the empty input tape. □

5 Conclusions

This paper provides a comprehensive solution to the distributed synthesis problem for alternating-time temporal logics. The synthesis problem is decidable if and only if the architecture is hierarchical. Our synthesis algorithm is uniformly applicable to all decidable architectures and all specification logics in the range from CTL to the alternating-time μ-calculus.

The central technical innovation is the treatment of nondeterministic implementations. We encode nondeterministic implementations as (deterministic) choice-set trees. This allows us to represent sets of strategies with tree automata and to distribute the global specification over the distributed architecture using standard automata transformations.

Nondeterministic implementations are also of interest if the specification is expressed in a standard branching-time logic like CTL*. In this case, nondeterminism means abstraction: details regarding the interaction with the external environment (including the user) can be omitted, since existential requirements can be demonstrated without immediately establishing the protocol. The resolution of the nondeterminism is moved to later design phases, where, in divide-and-conquer fashion, only a single nondeterministic component needs to be considered at a time.

References

1. Church, A.: Logic, arithmetic and automata. In: Proc. 1962 Intl. Congr. Math., Upsala (1963) 23–25
2. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Proc. IBM Workshop on Logics of Programs, Springer-Verlag (1981) 52–71
3. Wolper, P.: Synthesis of Communicating Processes from Temporal-Logic Specifications. PhD thesis, Stanford University (1982)
4. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable concurrent program specifications. In: Proc. 16th Int. Colloquium on Automata, Languages and Programming. Volume 372 of Lecture Notes in Computer Science., Springer-Verlag (1989) 1–17
5. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In ACM, ed.: Proc. POPL, New York, NY, USA, ACM Press (1989) 179–190
6. Kupferman, O., Vardi, M.Y.: μ-calculus synthesis. In: Proc. MFCS, Springer-Verlag (2000) 497–507

7. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proc. FOCS, IEEE Computer Society Press (1990) 746–757
8. Kupferman, O., Vardi, M.Y.: Synthesizing distributed systems. In: Proc. LICS, IEEE Computer Society Press (2001) 389–398
9. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proc. LICS, IEEE Computer Society Press (2005) 321–330
10. Kremer, S., Raskin, J.F.: A game-based verification of non-repudiation and fair exchange protocols. Journal of Computer Security **11**(3) (2003) 399–430
11. Mahimkar, A., Shmatikov, V.: Game-based analysis of denial-of-service prevention protocols. In: IEEE Computer Security Foundations Workshop. (2005) 287–301
12. Kremer, S.: Formal Analysis of Optimistic Fair Exchange Protocols. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium (2003)
13. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5) (2002) 672–713
14. van Drimmelen, G.: Satisfiability in alternating-time temporal logic. In: Proc. LICS, IEEE Computer Society Press (2003) 208–217
15. Schewe, S., Finkbeiner, B.: The alternating-time μ-calculus and automata over concurrent game structures. In: Proc. CSL, Springer-Verlag (2006) 591–605
16. Wilke, T.: Alternating tree automata, parity games, and modal μ-calculus. Bull. Soc. Math. Belg. **8**(2) (2001)
17. Kupferman, O., Vardi, M.Y.: Church's problem revisited. The bulletin of Symbolic Logic **5**(2) (1999) 245–263
18. Finkbeiner, B., Schewe, S.: Semi-automatic distributed synthesis. In: Proc. ATVA, Springer-Verlag (2005) 263–277
19. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, McNaughton and Safra. Theor. Comput. Sci. **141**(1-2) (1995) 69–107
20. Bhat, G., Cleaveland, R.: Efficient model checking via the equational μ-calculus. In: Proc. LICS, IEEE Computer Society Press (1996) 304–312
21. de Alfaro, L., Henzinger, T.A., Majumdar, R.: From verification to control: Dynamic programs for omega-regular objectives. In: Proc. LICS, IEEE Computer Society Press (2001) 279–290