Coordination Logic[⋆]

Bernd Finkbeiner[1] and Sven Schewe[2]

[1] Universität des Saarlandes
[2] University of Liverpool

Abstract. We introduce Coordination Logic (CL), a new temporal logic that reasons about the interplay between behavior and informedness in distributed systems. CL provides a logical representation for the distributed realizability problem and extends the game-based temporal logics, including the alternating-time temporal logics, strategy logic, and game logic, with quantification over strategies under incomplete information. We show that the structure in CL that results from the nesting of the quantifiers is sufficient to guarantee the decidability of the logic and at the same time general enough to subsume and extend all previously known decidable cases of the distributed realizability problem.

1 Introduction

An intriguing aspect of the design of a distributed system is the interplay between behavior and informedness: the *behavior* of a distributed system is determined by the combination of local decisions, made by processes that only have *partial information* about the system state. To ensure that the behavior of the full system is correct, the processes must carefully coordinate so that each process obtains the information needed for the right decision.

Historically, specification languages for distributed systems, in particular the temporal logics, have focused on specifying the acceptable behavior and ignored the question which level of informedness is needed to implement it. This is not surprising, because the main application of these logics has been verification, where one proves that the behavior of a complete system, with all implementation decisions already made, is correct. More recently, however, a lot of attention has shifted towards analyzing distributed systems at early design stages, where one has requirements but not yet a complete implementation [1,2,3,4,5,6,7,8,9]. At such a point in the design process, one is interested in *realizability* [6,7,8,9] (is there an implementation that satisfies the requirements?) and other *game-based* properties [1,2,3,4,5], such as: can a given coalition of processes accomplish a certain goal against all possible actions of the other processes?

In the game-based setting, the connection between behavior and information becomes crucially important, because the strategic capabilities of a process strongly depend on its informedness. Realizability and other game-based properties have therefore been investigated with respect to specific *system architectures*, that is, graphs of the communication topology, where the nodes represent

[⋆] This work was supported by DFG TR AVACS and by EPSRC grant EP/H046623/1.

A. Dawar and H. Veith (Eds.): CSL 2010, LNCS 6247, pp. 305–319, 2010.

processes and edges represent communication links. Unfortunately, very few system architectures can be analyzed algorithmically: even for simple architectures, like the two-process architecture of a distributed arbiter, the realizability problem for the standard temporal logics LTL, CTL, and CTL*, as well as the model checking problems for the game-based extensions, including the alternating-time temporal logics [1,4], strategy logic [2], and game logic [1], are undecidable. Most practical approaches therefore concentrate on the simple but unrealistic special case where all processes have complete information.

In this paper, we re-investigate the interplay of behavior and informedness from a logical perspective. We introduce *Coordination Logic (CL)*, the first specification language that defines behavior and informedness *within* the logic instead of by referring to an external system architecture. CL uses two types of variables. *Coordination variables* represent knowledge, directly observed from the external environment or obtained through coordination with other system processes. *Strategy variables* encode the result of strategic choices that are made based on the values of the coordination variables. Either type of information can be hidden from other processes by quantification. As we will see, CL is a decidable logic that is not only sufficiently expressive to encode all queries over system architectures for which the game-based logics are decidable, but can additionally, because behavior and informedness are represented in the same logical framework, answer many queries over systems with otherwise undecidable architectures.

We motivate CL with a simple example. Consider the "statement" functionality of an automated banking terminal, where the user can choose to either show the statement on the screen or get a printout. We model this system with three coordination variables, a for account number, s for screen, p for printout, and two strategy variables, d for data, modeling the content of the statement sent from a central server to the terminal, and o for output, modeling the terminal's output. CL includes linear-time temporal logic (LTL) as a sublogic for the specification of behavioral requirements, so let us assume that the functionality of the terminal has been specified by an LTL formula φ. The CL specification now precisely identifies what information is visible to the terminal and what information is visible to the server. For example, the formula $\Finv a, s, p\, \exists d, o\, \varphi$ specifies that the functionality specified in φ can be realized by communicating the full information to the server: the existential quantification $\exists d, o$ over the strategy variables d and o expresses the existence of a strategy for the server and the terminal; the existential quantification $\Finv a, s, p$ over the coordination variables a, s, and p introduces the information on which the decisions on d and o are based. Clearly, however, full information is not needed in this example: the server does not need to know whether the client wants the statement on the screen or on the printer. This is expressed by the alternative specification $\Finv a\, \exists d\, \Finv s, p\, \exists o\, \varphi$, which hides the coordination variables s and p from the strategy variable d. The example can easily be extended to systems with multiple terminals, resulting in the conjunction $\bigwedge_{i=1,\ldots,n} \Finv a_i\, \exists d_i\, \Finv s_i, p_i\, \exists o_i\, \varphi_i$. The resulting star topology (many terminals communicate with a single server) is an example of a system architecture where game-based properties are in general undecidable [9]; the representation as a CL formula reveals, however, that this particular query is in fact decidable.

The remainder of the paper is structured as follows. After reviewing some basic notation on trees and tree automata in Section 2, we formally define CL in Section 3. Section 4 is devoted to a thorough study of the expressiveness of CL: we show that CL subsumes the usual linear and branching-time logics LTL, CTL, and CTL*, as well as the game-based logics ATL and ATL* [1], strategy logic [2], and game logic [1]. We furthermore show that the *distributed realizability problem* [6,7,8,9] can be encoded in CL for all system architectures where the problem is decidable. In fact, we show that CL makes it easier to identify decidable cases. For example, in the special case of local specifications, where the LTL specification consists of a separate conjunct for each process, the doubly flanked pipeline architecture, in which the otherwise least informed process has an additional input, is also decidable [10]. We give a new, simple proof of the decidability of doubly-flanked pipelines and local specifications by encoding the realizability problem as a CL formula.

Related work. There is a rich literature on game-based extensions of temporal logic, including the alternating-time temporal logics [1,4], strategy logic [2], and game logic [1]. Model checking algorithms for these logics typically assume that all processes are fully informed; however, versions with incomplete information have been defined, in particular for the alternating-time temporal logics [1,11]. Unlike CL, none of these logics combine the specification of behavior and informedness in the same formula.

There are several variations of the alternating-time temporal logics that are orthogonal to our study of the interplay of behavior and informedness. Alternating-time temporal epistemic logic [12] extends strategic properties with knowledge modalities. Hence, while CL specifies the existence of strategies that are *based* on incomplete information, this logic specifies the existence of strategies to *obtain* certain knowledge. Since the strategies itself are based on full information, actions that require certain knowledge must be restricted individually with explicit knowledge preconditions. Other notable variations of the alternating-time temporal logics include the extension to strategy contexts [13,5] and the restriction to bounded resources [3,5].

Our decision procedure for CL builds on standard automata constructions used in synthesis algorithms for distributed systems, in particular for pipeline and, more generally, weakly-ordered system architectures [6,7,8,9]. Since distributed realizability can be encoded in CL, our decision procedure subsumes the decidable realizability problems. Moreover, since we study behavior and informedness in a common logical representation, new cases can be identified as decidable simply by encoding the queries in CL.

2 Preliminaries

Trees. We use trees as a representation for strategies and computations. As usual, a (full) *tree* is given as the set Υ^* of all finite words over a given set of directions Υ. For given finite sets Σ and Υ, a *Σ-labeled Υ-tree* is a pair $\langle \Upsilon^*, l\rangle$ with a labeling function (or strategy) $l : \Upsilon^* \to \Sigma$ that maps every node of Υ^* to a letter of Σ. For a set $\Xi \times \Upsilon$ of directions and a node $x \in (\Xi \times \Upsilon)^*$, $\mathit{hide}_\Upsilon(x)$ denotes the node in Ξ^* obtained by replacing (ξ, υ) by ξ in each letter of x.

For a $\Sigma \times \Xi$-labeled Υ-tree $\langle \Upsilon^*, l\rangle$ we define the Ξ-projection of $\langle \Upsilon^*, l\rangle$, denoted by $\mathit{proj}_\Xi(\langle \Upsilon^*, l\rangle)$, as the Σ-labeled Υ-tree $\langle \Upsilon^*, \mathrm{pr}_1 \circ l\rangle$. That is, $\mathit{proj}_\Xi(\langle \Upsilon^*, l\rangle)$ is obtained from $\langle \Upsilon^*, l\rangle$ by projecting away the Ξ-part from the label in every node of Υ^*.

For a Σ-labeled Ξ-tree $\langle \Xi^*, l\rangle$ we define the Υ-widening of $\langle \Xi^*, l\rangle$, denoted by $\mathit{wide}_\Upsilon(\langle \Xi^*, l\rangle)$, as the Σ-labeled $\Xi \times \Upsilon$-tree $\langle (\Xi \times \Upsilon)^*, l'\rangle$ with $l'(x) = l(\mathit{hide}_\Upsilon(x))$. By abuse of notation, we use $\mathit{wide}_\Upsilon(l)$ for l'. In $\mathit{wide}_\Upsilon(\langle \Xi^*, l\rangle)$, nodes that are indistinguishable for someone who cannot observe Υ—that is, nodes x, y with $\mathit{hide}_\Upsilon(x) = \mathit{hide}_\Upsilon(y)$—have the same label.

If $\Xi = 2^X$ and $\Upsilon = 2^Y$ are power-sets of disjoint sets X and Y, we identify $\Xi \times \Upsilon$ with $2^{X \dot\cup Y}$, and use the widening operator wide_Υ accordingly. Let X and Y be sets with $Z = X \cup Y$, and let M and N be disjoint set. Then we denote with $\langle (2^Z)^*, l_M : (2^Z)^* \to 2^M\rangle \dot\cup \langle (2^Z)^*, l_N : (2^Z)^* \to 2^N\rangle$ the $2^{M \dot\cup N}$-labeled tree $\langle (2^Z)^*, l\rangle$ with $l(x) = l_M(x) \dot\cup l_N(x)$, and use $\langle (2^X)^*, l\rangle \oplus \langle (2^Y)^*, l'\rangle = \mathit{wide}_{2^{Z \smallsetminus X}}(\langle (2^X)^*, l\rangle) \dot\cup \mathit{wide}_{2^{Z \smallsetminus Y}}(\langle (2^Y)^*, l'\rangle)$.

For a Σ-labeled Υ-tree $\langle \Upsilon^*, l\rangle$ and a word $x \in \Upsilon^*$, we denote with $\langle \Upsilon^*, l\rangle|x$ the sub-tree rooted in x, that is, the tree $\langle \Upsilon^*, l'\rangle$ with $l'(y) = l(x \cdot y)$.

Automata. An (alternating parity) automaton $\mathcal{A} = (\Sigma, Q, q_0, \delta, \alpha)$ runs on Σ-labeled Υ-trees (for a predefined finite set Υ of directions). Q denotes a finite set of states, $q_0 \in Q$ denotes a designated initial state, δ denotes a transition function $\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times \Upsilon_\varepsilon)$ for $\Upsilon_\varepsilon = \Upsilon \dot\cup \{\varepsilon\}$, and $\alpha : Q \to C$ is a function that maps the states of $\mathcal{A}$ to a finite set of colors $C \subset \mathbb{N}$. If $C = \{0, 1\}$, we call $\mathcal{A}$ a co-Büchi automaton.

A run tree on a given 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 a node n with a label (q, x) and a set of children $\mathit{child}(n)$, the labels of these children have the following properties:

- for all $m \in \mathit{child}(n)$: the label of m is $(q_m, x \cdot \upsilon_m)$, $q_m \in Q, \upsilon_m \in \Upsilon_\varepsilon$ such that (q_m, υ_m) is an atom of $\delta(q, l(x))$, and
- the set of atoms defined by the children of n satisfies $\delta(q, l(x))$.

A path is accepted if the highest color appearing infinitely often is even. A run tree is *accepting* if all its paths are accepted, and a Σ-labeled Υ-tree is accepted by $\mathcal{A}$ if it has an accepting run tree. The set of trees accepted by an automaton $\mathcal{A}$ is called its *language* $\mathcal{L}(\mathcal{A})$.

The acceptance of a tree can also be viewed as the outcome of a game, where player *accept* chooses, for every pair $(q, x) \in Q \times \Upsilon^*$, a set of atoms that satisfy $\delta(q, l(x))$, and player *reject* chooses one of these atoms, which is executed. The input tree is accepted iff player *accept* has a strategy to enforce a path that satisfies the parity condition.

Theorem 1 (narrowing). *[14] For an automaton $\mathcal{A} = (\Sigma, Q, q_0, \delta, \alpha)$ over $\Xi \times \Upsilon$-trees, we can build an automaton $\mathcal{A}_n = (\Sigma, Q, q_0, \delta', \alpha)$ that accepts a Σ-labeled tree $\langle \Upsilon^*, l\rangle$ iff $wide_\Xi(\langle \Upsilon^*, l\rangle)$ is accepted by $\mathcal{A}$.* □

Theorem 2 (projection). *[15] For an automaton $\mathcal{A} = (\Sigma \times \Xi, Q, q_0, \delta, \alpha)$ over Υ-trees, we can build (1) an automaton $\mathcal{A}^b_w = (\Sigma, Q', q_0, \delta', \alpha')$ that accepts a Σ-labeled tree $\langle \Upsilon^*, l\rangle$ iff* some *$\Sigma \times \Xi$-labeled tree $\langle \Upsilon^*, l'\rangle$ that satisfies $\langle \Upsilon^*, l\rangle = proj_\Xi(\langle \Upsilon^*, l'\rangle)$ is accepted by $\mathcal{A}$, and (2) an automaton $\mathcal{A}^b_s = (\Sigma, Q', q_0, \delta', \alpha')$ that accepts a Σ-labeled tree $\langle \Upsilon^*, l\rangle$ iff* all *$\Sigma \times \Xi$-labeled trees $\langle \Upsilon^*, l'\rangle$ that satisfy $\langle \Upsilon^*, l\rangle = proj_\Xi(\langle \Upsilon^*, l'\rangle)$ are accepted by $\mathcal{A}$.* □

Note that, for technical convenience in the upcoming proofs, we assume that the initial state of $\mathcal{A}$ and the initial state of the respective transformed automaton are the same.

We call an automaton a *word automaton* if the set of directions is singleton, and *universal* if δ consists only of conjunctions. By abuse of notation, we interpret $\delta(q)$ as the set of its conjuncts for universal word automata.

3 Coordination Logic

Syntax. CL formulas are defined over two types of variables: coordination variables $\mathcal{C}$, and strategy variables $\mathcal{S}$. The operators of CL consist of the usual LTL operators Next $\bigcirc$, Until $\mathcal{U}$, and the dual Until $\overline{\mathcal{U}}$, as well as the new *subtree quantifiers* $\exists C \exists s.\varphi \mid \exists C \forall s.\varphi$. The syntax of CL is given by the grammar

$$\varphi ::= x \mid \neg x \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi\, \mathcal{U}\, \varphi \mid \varphi\, \overline{\mathcal{U}}\, \varphi \mid \exists C \exists s.\varphi \mid \exists C \forall s.\varphi,$$

where $x \in \mathcal{C} \dot{\cup} \mathcal{S}$, $C \subseteq \mathcal{C}$, and $s \in \mathcal{S}$.

We call a formula *well-formed* if the sets of coordination variables that occur under different $\exists$-operators are pairwise disjoint, and the same strategy variable is not introduced more than once. In the following we assume that all formulas are well-formed. (Well-formedness can be ensured by a suitable renaming.)

Note that we combine, for technical convenience and to emphasize the close connection between coordination and strategy variables, the introduction of both types of variables into a single subtree quantifier. We use $\forall s.\ \varphi$ and $\exists s.\ \varphi$ as abbreviations for $\exists \emptyset \forall s.\ \varphi$ and $\exists \emptyset \exists s.\ \varphi$, respectively, and $\exists C.\ \varphi$ as an abbreviation for $\exists C \forall s.\ \varphi$ (or, likewise, $\exists C \exists s.\ \varphi$), where s is a fresh strategy variable (which in particular does not occur in φ). We also use the standard abbreviations $true \equiv x \vee \neg x$, $false \equiv x \wedge \neg x$, $\diamondsuit\, \varphi \equiv true\, \mathcal{U}\, \varphi$, and $\square\, \varphi \equiv false\, \overline{\mathcal{U}}\, \varphi$.

Semantics. Coordination variables provide strategy variables with the information required for their decisions. Following the structure of a formula, a bound coordination variable c is *visible* to a bound strategy variable s, if s is in the scope of the subtree quantifier that introduced c. We denote the set of bound coordination variables that are visible to s by $scope(s)$. The free strategy variables, also called *atomic propositions*, are denoted by Π. For all atomic propositions $p \in \Pi$ it holds that $scope(p) = \emptyset$.

The set of free coordination variables is denoted by $\mathcal{F}$. Free coordination variables are visible to all strategy variables. We call the set of coordination variables visible to a strategy variable s the *scope* of s, denoted $Scope(s) = scope(s) \dot{\cup} \mathcal{F}$. By abuse of notation, we use $scope(S) = \bigcup_{s \in S} scope(s)$ and $Scope(S) = \bigcup_{s \in S} Scope(s)$ for a set $S \subseteq \mathcal{S}$ of strategy variables, and $Scope(\exists C Q s.\varphi) = Scope(s)$, for $Q \in \{\exists, \forall\}$.

The meaning of a strategy variable s is a *strategy* $f_s : (2^{Scope(s)})^* \to 2^{\{s\}}$, i.e., a mapping from the information available to s, which consists of the history of the valuations of the coordination variables in the scope of s, to a (Boolean) valuation of s.

For a subset $S \subseteq \mathcal{S}$ of the strategy variables, we call the tree $\mathcal{T} = \bigoplus_{s \in S} \langle (2^{Scope(s)})^*, f_s \rangle$ defined by their joint valuation a *frame* over S. CL formulas are interpreted over frames $\mathcal{T} = \bigoplus_{p \in \Pi} \langle (2^{\mathcal{F}})^*, f_p \rangle$ over the atomic propositions, called *computation trees*.

As usual, a path in the tree $\mathcal{T}$ is an ω-word $\sigma = \sigma_0 \sigma_1 \sigma_2 \ldots \in (2^{\mathcal{F}})^\omega$, and we denote the related labeled path by $\sigma^{\mathcal{T}} = (l(\varepsilon) \cup \sigma_0)(l(\sigma_0) \cup \sigma_1)(l(\sigma_0\sigma_1) \cup \sigma_2)(l(\sigma_0\sigma_1\sigma_2) \cup \sigma_3) \ldots \in (2^{\mathcal{F} \cup \Pi})^\omega$.

A tree $\mathcal{T}$ satisfies a CL formula φ, denoted by $\mathcal{T} \models \varphi$, iff all paths satisfy φ, i.e., $\forall \sigma \in (2^{\mathcal{F}})^\omega.\ \sigma, 0 \models^{\mathcal{T}} \varphi$, where the satisfaction of a CL formula φ on a path σ at position $i \geq 0$, denoted by $\sigma, i \models^{\mathcal{T}} \varphi$, is defined inductively as follows:

- for the strategy and coordination variables $x \in S \dot{\cup} \mathcal{C}$,
  - $\sigma, i \models^{\mathcal{T}} x :\Leftrightarrow x \in \sigma^{\mathcal{T}}(i)$, and
    $\sigma, i \models^{\mathcal{T}} \neg x :\Leftrightarrow x \notin \sigma^{\mathcal{T}}(i)$ for all $x \in \mathcal{S} \cup \mathcal{C}$;
- for the boolean connectives, where φ and ψ are CL formulas,
  - $\sigma, i \models^{\mathcal{T}} \varphi \vee \psi :\Leftrightarrow \sigma, i \models^{\mathcal{T}} \varphi$ or $\sigma, i \models^{\mathcal{T}} \psi$, and
  - $\sigma, i \models^{\mathcal{T}} \varphi \wedge \psi :\Leftrightarrow \sigma, i \models^{\mathcal{T}} \varphi$ and $\sigma, i \models^{\mathcal{T}} \psi$;
- for the temporal path operators, where φ and ψ are CL formulas,
  - $\sigma, i \models^{\mathcal{T}} \bigcirc \varphi :\Leftrightarrow \sigma, i+1 \models^{\mathcal{T}} \varphi$,
  - $\sigma, i \models^{\mathcal{T}} \varphi\, \mathcal{U}\, \psi :\Leftrightarrow \exists n \geq i.\ \sigma, n \models^{\mathcal{T}} \psi$ and $\forall m \in \{i, \ldots, n-1\}.\ \sigma, m \models^{\mathcal{T}} \varphi$,
  - $\sigma, i \models^{\mathcal{T}} \varphi\, \overline{\mathcal{U}}\, \psi :\Leftrightarrow \forall n \geq i.\ \sigma, n \models^{\mathcal{T}} \psi$ or
    $\exists n \geq i.\ \sigma, n \models^{\mathcal{T}} \varphi$ and $\forall m \in \{i, \ldots, n\}.\ \sigma, m \models^{\mathcal{T}} \psi$;
- for the subtree quantifiers, where $C \subseteq \mathcal{C}$, $s \in \mathcal{S}$, and φ is a CL formula,
  - $\sigma, i \models^{\mathcal{T}} \exists C \exists s.\varphi :\Leftrightarrow \exists f : (2^{Scope(s)})^* \to 2^{\{s\}}.\ (\mathcal{T}|\sigma_i) \oplus \langle (2^{Scope(s)})^*, f \rangle \models \varphi$,
    where σ_i is the initial sequence of length i of σ,
  - $\sigma, i \models^{\mathcal{T}} \exists C \forall s.\varphi :\Leftrightarrow \forall f : (2^{Scope(s)})^* \to 2^{\{s\}}.\ (\mathcal{T}|\sigma_i) \oplus \langle (2^{Scope(s)})^*, f \rangle \models \varphi$.

4 Expressive Power of Coordination Logic

Coordination logic is sufficiently expressive to subsume the model-checking problem for all other popular temporal logics, in particular for LTL, CTL, CTL*, ATL, ATL*, and strategy logic. Moreover, it can also express the satisfiability of these logics, as long as the branching degree of the models is fixed and can therefore be encoded using a finite set of variables. For logics like ATL*, where models of fixed branching degree are known to be sufficient [16], the satisfiability problem can thus be expressed in CL.

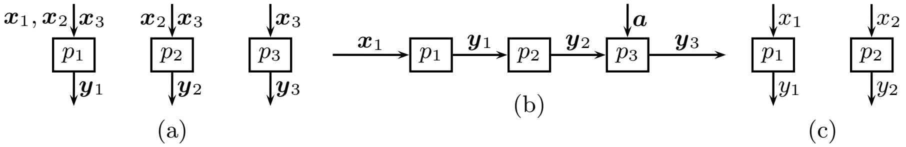

Fig. 1. Distributed architectures

Proposition 1. *The decision problem for strategy logic, game logic, and ATL* and its sub-logics over models with a fixed branching degree can be encoded in CL.*

The distributed realizability problem is defined with respect to a system architecture, which is a directed finite graph whose nodes correspond to processes and whose edges are labeled with sets of variables: each process can read the variables on incoming edges and write to variables on the outgoing edges. There is a distinguished process, called the *environment process*, that provides the input to the system. Given such a system architecture, the *distributed realizability problem* [6,7,8,9] checks for the existence of a set of strategies, one for each process, such that the combination of these strategies satisfies the specification. We now describe an encoding of the realizability problem in CL for the two known decidable cases, the weakly-ordered architectures and the doubly-flanked pipelines. We begin with a simple special case of the weakly-ordered architectures, the linear architectures (cf. Figure 1a).

Realizability in linear architectures. In a *linear* architecture, each process reads only inputs from the external environment, not any outputs of other processes. Furthermore, we assume that the sets of variables visible to the processes are pairwise comparable (cf. Figure 1a) and, hence, define a linear 'informedness' pre-order on the output variables.

In the following, we will reuse the variables from the architecture as coordination and strategy variables in CL, denoting a vector $x_1, x_2, \ldots, x_i$ of variables by a single symbol $\boldsymbol{x}$. The realizability of an LTL formula φ in a linear architecture can be encoded as the CL formula $\text{⅃}\boldsymbol{x}_n\exists\boldsymbol{y}_n \ldots \text{⅃}\boldsymbol{x}_2\exists\boldsymbol{y}_2\text{⅃}\boldsymbol{x}_1\exists\boldsymbol{y}_1\varphi$, where the strategy variables $\boldsymbol{y}_i$ represent the output of the i-th best informed processes, and the coordination variables $\boldsymbol{x}_i$ represent the inputs from the environment that are available to the ith best informed processes, but not the $i+1$st best informed processes.

Realizability in weakly-ordered architectures. Weakly-ordered architectures generalize linear architectures by allowing the processes to also read the output of other processes, while still requiring that ordering the processes according to their informedness results in a linear pre-order on the output variables. The realizability of temporal specifications in weakly-ordered architectures is decidable. In fact, the class of weakly-ordered architectures consists exactly of those architectures for which the realizability problem is decidable [9].

In a weakly-ordered architecture, we can assume w.l.o.g. that each process reads all inputs of less-informed processes directly as its own input [9]. In our encoding, we introduce additional coordination variables that simulate the input for the processes and then restrict the LTL formula to those paths where the true input and the mock input we introduced coincide. We obtain the CL formula $\dashv\boldsymbol{x}_n\exists\boldsymbol{y}_n\ldots\dashv\boldsymbol{x}_2\exists\boldsymbol{y}_2\dashv\boldsymbol{x}_1\exists\boldsymbol{y}_1(\Box\,\boldsymbol{x}^{mock}=\boldsymbol{y}^{real})\rightarrow\varphi$, where $\boldsymbol{x}_i$ are the (mock) input variables, and $\Box\,\boldsymbol{x}^{mock}=\boldsymbol{y}^{real}$ is the restriction to the paths in which the mock input corresponds to the true input.

Proposition 2. *The realizability problem for weakly-ordered architectures can be encoded in CL.*

This encoding also demonstrates the conciseness of CL: The encoding consists only of the LTL specification from the distributed realizability problem, and at most four copies of each variables. Deciding CL is thus at least as expensive as the realizability problem for weakly-ordered architectures, which is nonelementary [7].

Proposition 3. *The decision and model checking problem of CL are non-elementary.*

Since CL is decidable (cf. Corollary 1 in Section 5), it is clear that the realizability problem of undecidable architectures cannot be encoded in CL, but this does by no means imply that CL is restricted to encoding problems for decidable architectures. An interesting example are doubly-flanked pipeline architectures, for which the realizability of local LTL specifications is decidable.

Local specifications and doubly-flanked pipelines. A specification is called *local* if it only reasons about the variables of a single process, and a doubly flanked pipelines is, as shown in Figure 1b, a pipelines in which the otherwise least informed process has an additional input. The realizability of local LTL specifications in doubly-flanked pipelines is decidable [10]. In our encoding, we divide the realizability problem into two parts: one for the pipeline (minus process p_n) and one for p_n, and then couple these problems logically. This can be done by extending both pipelines (the one of length $n-1$ and the one of length 1) by a shadow process p_e which controls a Boolean variable e and reads a mock input $\boldsymbol{x}_n$ reflecting $\boldsymbol{y}_{n-1}$ (which is regarded as a true environment input for the short pipeline from the right); if ψ is the local specification for the rightmost process, and φ the conjunction over the specifications of the remaining processes, then we can simply strengthen φ to $\varphi\wedge\Box\,e$, forcing e to be true on all possible runs of $\boldsymbol{y}_{n-1}$, and weaken ψ to $\Box\,e\rightarrow\psi$, which restricts ψ to paths marked as existing. Now, both of these sub-specifications start with $\dashv\boldsymbol{x}_n\exists e$, and it is easy to see that applying a conjunction after this operator provides the right level of light entanglement between the two sub-specifications.

In case of the example architecture from Figure 1b, this specification reads $\dashv\boldsymbol{x}_3\exists e\big(\dashv\boldsymbol{x}_2\exists\boldsymbol{y}_2\dashv\boldsymbol{x}_1\exists\boldsymbol{y}_1(\Box\,\boldsymbol{x}_2=\boldsymbol{y}_1\wedge\boldsymbol{x}_3=\boldsymbol{y}_2)\rightarrow(\varphi\wedge\Box\,e)\big)\wedge\dashv\boldsymbol{a}\exists\boldsymbol{y}_3(\Box\,e\rightarrow\psi)$.

Proposition 4. *The distributed realizability problem for doubly flanked pipelines can be encoded in CL.*

In fact, this encoding can easily be generalized to larger classes of specifications. In the full version of this paper [17] we describe such an extension.

5 Automata-Based Decision Procedure

Our decision procedure for coordination logic draws from the rich tool box of ω-automata transformations available in the literature for satisfiability checking and synthesis. As a preparation for the development of the decision procedure, we first rephrase the semantics for coordination logic as a model-checking game, which checks the correctness of $\mathcal{T} \models \varphi$.

Model-Checking Game. For a CL formula φ, we denote with $dqsf(\varphi)$ the set of direct quantified sub-formulas defined by $dqsf(\exists CQs.\varphi) = \widehat{dqsf}(\varphi)$ for $\widehat{dqsf}(x) = \widehat{dqsf}(\neg x) = \emptyset$ for $x \in \mathcal{C} \cup \mathcal{S}$, $\widehat{dqsf}(\varphi \circ \psi) = \widehat{dqsf}(\varphi) \cup \widehat{dqsf}(\psi)$ for $\circ \in \{\wedge, \vee, \mathcal{U}, \overline{\mathcal{U}}\}$, $\widehat{dqsf}(\bigcirc \varphi) = \widehat{dqsf}(\varphi)$, and $\widehat{dqsf}(\exists CQs.\varphi) = \{\exists CQs.\varphi\}$.

With $\widehat{\psi}$, we denote the formula derived from ψ by replacing its direct quantified sub-formulas $dqsf(\varphi)$ by variables that reflect their correctness.

A model-checking game can, starting with the candidate $\mathcal{T}$ under consideration, expand $\mathcal{T}$ along the specification. Acceptance is determined by a game between two players, *accept* and *reject*. For unquantified formulas, the model-checking for $\mathcal{T} \models \varphi$ starts in a state $(\mathcal{T}, \varphi)$ and proceeds as follows:

1. player *accept* guesses the correctness of the state formulas $dqsf(\varphi)$, expanding the labeling function of $\mathcal{T}$ to $\mathcal{T}_1$,
2. player *reject* either (a) chooses an infinite path σ in $\mathcal{T}_1$ and test if it satisfies $\widehat{\varphi}$ and stop the game, or she (b) chooses a finite path π in $\mathcal{T}_1$ and a formula $\exists CQs.\psi \in \widehat{\varphi}$ that is marked valid in this position in $\mathcal{T}_1$, and proceeds in $\mathcal{T}_2 = \mathcal{T}_1|\pi$,
3. $\mathcal{T}_2$ is widened by 2^C to $\mathcal{T}_3 = wide_{2^C}(\mathcal{T}_2)$
4. for a formula $\exists C\exists s.\psi$ player accept, and for a formula $\exists C\forall s.\psi$ player reject extends the labeling of $\mathcal{T}_3$ by a valuation of s to $\mathcal{T}_4$, and the game proceeds in (1) from state $(\mathcal{T}_4, \psi)$.

For quantified formulas $\varphi = \exists CQs.\psi$, we choose initially $\mathcal{T}_3 = \mathcal{T}$ and start in Step (3).

Player *accept* wins if the path evaluated at the end of the game in Step (2) satisfies $\widehat{\varphi}$, while player reject wins otherwise. (Note that the game goes down in the formula tree in each round, and player *reject* must eventually choose to validate a path.)

Proposition 5. *The model-checking game is won by player accept iff $\mathcal{T} \models \varphi$.*

Proof. The game simply follows the structural definition of the semantics.

'⇒': Let $\mathcal{T} \models \varphi$. For the choices in Step (1), we can assume that player accept uses a perfect oracle, whose choice can be challenged by player reject. By the perfect choices of the oracle, player accept won the valuation of any path player reject chose in Step (2a), which might leave player reject with choosing a sub-formula that is claimed to be valid—and is therefore valid as the oracle is perfect—in Step (2b). By the definition of validity, this means that any (for a universally quantified sub-formula $\exists C \forall s.\psi$) or some (for an existentially quantified sub-formula $\exists C \exists s.\psi$) extension of the proper widening of the tree contains only paths that satisfy ψ, which is reflected by Steps (3) and (4). This argument can then be repeated with the chosen sub-formula, until the a sub-formula φ with an empty set $dqsf(\varphi) = \emptyset$ of direct quantified sub-formulas is reached.

'$\Leftarrow$': Let $\mathcal{T} \not\models \varphi$. For the choices player *accept* makes in Step (1), player *reject* uses a perfect oracle to find a position and direct quantified sub-formula, such that the sub-formula is claimed to be true. If no such position exists, there must be a trace that contradicts φ by our assumption, and player *reject* wins. (Note that claiming that a satisfied direct quantified sub-formula does not hold cannot help the acceptance of a path, because our specification is in positive form.) If such a position exists, the choice of the perfect oracle shows that some (for a sub-formula $\exists C \forall s.\psi$) or all (for a sub-formula $\exists C \forall s.\psi$) extensions of the proper widening of the tree is contains a path that does not satisfy ψ. This argument can again be repeated with the chosen sub-formula, until a sub-formula ψ with an empty set $dqsf(\varphi) = \emptyset$ of direct quantified sub-formulas is reached.

To bridge the remaining gap between this game and working with trees, we can obviously change the second step of the game to checking all paths and, for each direct quantified sub-formula and each position in the tree, constructing a copy of the tree. For each position in the tree, the respective proof obligation is handed down to the copies spawned for the direct quantified sub-formulas that are marked valid (but not for those marked invalid), and the procedure is continued for each copy as if it was chosen in the game described above.

This adjustment indicates the relation of the CL model-checking problem with current synthesis procedures [6,7,8,9] and outlines the required changes. In distributed synthesis, causal bound variables are usually existentially quantified and occur in a prenex. For such prenex quantifiers, the truth of sub-formulas is only relevant in the root ε, and the set of direct quantified sub-formulas is singleton or empty. Hence, while we need to expand the tree by copying the respective sub-tree and perform widening operations on all copies individually for CL, it suffices for current synthesis procedures to cope with widening operations for the complete tree.

Our automata-based decision procedure performs the inverse to the functions occurring in the adjusted acceptance game: narrowing [14] as the inverse of widening (Step 3 of the adjusted game), friendly and hostile projection [15] of a strategy as the inverse of friendly and hostile causal choice (Step 1 and 4), and re-routing the test-automaton for correctness of a direct quantified sub-formula from the copy to its blueprint (Step 2b). We therefore have to introduce a more intricate tree structure, to reason about sub-trees instead of full trees, and to re-route intermediate test-automata.

Down-trees. Our construction is based on *down-trees* (cf. Figure 2), which follow the structure of a CL specification. Down-trees are trees of trees that refer to a quantified sub-formula. Down-trees can be viewed as the outcome of the model checking game, where the operations are applied in all places at the same time, and serve as a structure to run an automata based technique that

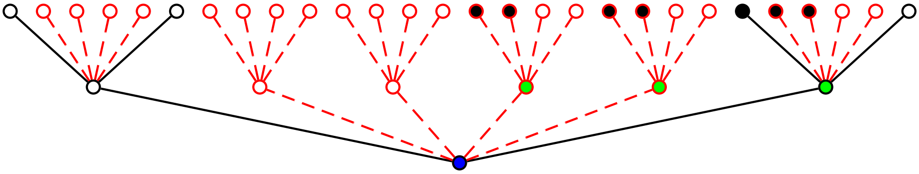

Fig. 2. Example down-tree: In a down tree, we start out with a slim tree that only branches by the valuations of the free communication variables, like the tree in black (full lines). This tree is labeled by the free strategy variables. An operator $\exists c \exists s$ spawns a new wider sub-tree (dashed lines) with a richer labeling function that additionally includes truth values of the bound strategy variable s. After projecting the new strategy variable s away, the new sub-tree is a *widening* of the sub-tree rooted in the respective node. That is, when restricted to the free strategy variables, the labeling of every path that is indistinguishable by the free coordination variables in the resulting tree is similar, as indicated by the blue-green-black path in the initial tree and its many copies of its ending sequences in the newly spawned sub-trees.

step-wise undoes the decisions in such a game. Viewed as a concurrent version of the game, the black (full) basic tree spawns siblings in each node, which are first widened, and whose label is then extended, using a friendly guess in case of existential quantifier, and a hostile guess in case of a universal quantifier. The automata-based construction introduced below puts a weak projection against the friendly guess, a strong projection against the hostile guess, and a narrowing operation against the widening.

Let Φ be the set of quantified sub-formulas of φ plus φ itself. We call a subset $\Psi \subseteq \Phi$ of Φ *sub-formula closed* if (1) $\psi' \in \Psi$ and $\psi \in dqsf(\psi') \cap \Phi$ imply $\psi \in \Psi$, and if (2) $\Psi \neq \emptyset$ implies $\varphi \in \Psi$. A formula $\psi \in \Psi$ is called *minimal* if none of its sub-formulas is in Ψ $(dqsf(\psi) \cap \Psi = \emptyset)$.

For each formula $\psi \in \Psi$, we introduce a set $D_\psi = \{\psi\} \times 2^{Scope(\psi)}$ of directions, and denote their union by $D_\Psi = \bigcup_{\psi \in \Psi} D_\psi$. We call a sequence $d = \delta_0\delta_1 \ldots \delta_n \in {D_\Psi}^*$ *falling* if, for all $i \geq 0$, $\delta_i \in D_\psi$ and $\delta_{i+1} \in D_{\psi'}$ implies that ψ' is a (not necessarily true) sub-formula of ψ. We call the set $\downarrow D_\Psi \subset {D_\Psi}^*$ of falling sequences in D_Ψ a *down-tree*, and denote, for a $\psi' \in \Psi$ and a $\psi \in dqsf(\psi') \cap \Psi$ that has no sub-formulas in Ψ $(dqsf(\psi) \cap \Psi = \emptyset)$, with $\downarrow D_\Psi^\psi$ the down-tree obtained when using $D_\psi = \{\psi\} \times 2^{Scope(\psi')}$ (that is, we intuitively hide the fresh information from the variable bound by the leading quantifier of ψ).

We call the formula ψ from the first projection of the last letter $\delta_n = (\psi, V)$ of $d = \delta_0\delta_1 \ldots \delta_n$ the *head* of a node d, and define φ to be the head of the empty word ε.

According to our demands, the labeling function for our down-trees has an amorphous co-domain that depends on the head of the node. A node with head ψ is labeled by a validity claim for all quantified super- and sub-formulas of ψ in Ψ (including ψ itself), the decisions of all bound strategy variables that are bound by a leading quantifier of these super- and sub-formulas, and the atomic propositions. We call down-trees with this amorphous co-domain Σ_Ψ-labeled down-trees. For the co-domain obtained by removing, for a minimal formula

$\exists CQs.\vartheta \in \Psi$, the $2^{\{s\}}$ part from the amorphous co-domain, we refer to Σ^s_Ψ-labeled trees.

For $\varphi \neq \psi$ and a down-tree $\downarrow D_\Phi$, we call a sub-trees $\mathfrak{D} = d \cdot D^*_\psi \subseteq \downarrow D_\Phi$, for which ψ is a true sub-formula of the head of $d \in D_\Phi$, a *ψ-tree* in $\downarrow D_\Phi$. We call $\mathfrak{D}$ in $\downarrow D_\Phi$ *reachable* if, for all $d' \cdot \delta \in \mathfrak{D}$, the following holds: for the head φ' of d' and the head ψ' of $d' \cdot \delta$, all quantified true sub-formulas of φ' that contain ψ' as their sub-formula are marked as valid in $l(d')$. ${D_\varphi}^*$ is the only φ-tree, and always reachable.

For a reachable ψ-tree $\mathfrak{D} = d \cdot D^*_\psi \subseteq \downarrow D_\Phi$ of a labeled down-tree $\langle \downarrow D_\Phi, l\rangle$, a path in $\mathfrak{D}$ is an ω-word $d, d \cdot (\psi, v_1), d \cdot (\psi, v_1) \cdot (\psi, v_2), \ldots$, and its trace is the ω-word $(l(d) \dot\cup v_1), (l(d) \dot\cup v_2), \ldots$. The reachable ψ-trees directly relate to the trees constructed in the adjusted model-checking game:

Observation 3. *For a given 2^Π-labeled $2^\mathcal{F}$-tree $\mathcal{T}$ and a given formula φ, the outcome of the adjusted acceptance game is a labeled down-tree intersected with the reachable trees.*

Player accept wins if, for all $\psi \in \Phi$, the trace of every labeled path of every reachable ψ-tree satisfies ψ.

We call a labeled down-tree where, for all $\psi \in \Phi$, the trace of every labeled path of every reachable ψ-tree satisfies ψ, *path accepting.*

Automata-Based Decision Procedure. Building on the above observation, we construct an automaton that accepts a labeled down-tree iff it is path accepting.

The simple path acceptance condition can be encoded in a universal co-Büchi automaton over labeled down-trees.

Lemma 1. *We can build a universal co-Büchi automaton that accepts a labeled down-tree iff it is path accepting.*

Construction: We first build, for every formula $\psi \in \Phi$, a universal co-Büchi automaton $\mathcal{U}_\psi = (\Sigma, Q_\psi, q_\psi, \delta_\psi, \alpha_\psi)$ for the claim-formula $\widehat{\psi}$ of ψ, using the standard translation for LTL. (The valuation of variables not in the language of $\mathcal{U}_\psi$ do not affect δ.) Using these automata as building blocks, we then build the universal automaton $\mathcal{U}^\varphi = (\Sigma, Q, \overline{\varphi}, \delta, \alpha)$, where Q consists of the disjoint union of the states of the individual $\mathcal{U}_\psi$, a fresh initial state $\overline{\varphi}$, and two fresh states, $\widetilde{\psi}$ and $\overline{\psi}$, for all other formulas $\psi \neq \varphi$ in Φ. All fresh states have color 0, while each other state $q \in Q_\psi$ keeps its color $\alpha(q) = \alpha_\psi(q)$.

The transition function δ is defined by

- $\delta(\overline{\psi}, \sigma) = (q_\psi, \varepsilon) \wedge \bigwedge_{\psi' \sigma \cap dqsf(\psi)} \delta(\widetilde{\psi'}, \varepsilon)$,
- $\delta(\widetilde{\psi}, \sigma) = \bigwedge_{d \in D_\vartheta} (\widetilde{\psi}, d)$ for $\psi \in dqsf(\vartheta)$ if $\psi \notin \sigma$ and
  $\delta(\widetilde{\psi}, \sigma) = \bigwedge_{d \in D_\vartheta} (\widetilde{\psi}, d) \wedge (\overline{\psi}, \varepsilon)$ otherwise, and
- $\delta(q, \sigma) = \bigwedge_{(\psi, v) \in D_\psi} \bigwedge_{q' \in \delta_\psi(q, \sigma \dot\cup v)} \big(q', (\psi, v)\big)$ for all $q \in Q_\psi$. □

The role of the fresh states is to ensure that checking is initiated at the root of exactly the reachable ψ-trees, and the transition functions from the last line ensure that the correct paths are checked for the correct properties from there.

We call the states in Q_ψ of $\mathcal{U}^\varphi$ *ψ-states*, the respective state q_ψ the *initial ψ-state*, and $\overline{\psi}$ and $\widetilde{\psi}$ *fresh states.*

Using this automaton as a starting point, we can construct an alternating automaton that accepts the models of a specification by successively applying the inverse operations of widening and guessing operations from the adjusted acceptance game.

The widening and projection operation for standard trees naturally extend to down-trees. For a Σ-labeled down-tree $\langle \downarrow D_\Psi, l\rangle$, $proj_{\Xi}(\langle \downarrow D_\Psi, l\rangle)$ denotes the Σ'-labeled down-tree $\langle \downarrow D_\Psi, l'\rangle$ that results from node-wise projecting Ξ, provided the co-domain for the node is $\Xi' \times \Xi$ for some Ξ'.

For a minimal formula $\psi = \dashv CQs.\vartheta'$ of Ψ, a Σ^s_Ψ-labeled down-tree $\mathcal{T} = \langle \downarrow D_\Psi, l\rangle$ is the ψ-widening of a Σ^s_Ψ-labeled down-tree $\mathcal{T}' = \langle \downarrow D^\psi_\Psi, l'\rangle$, denoted $wide^{D_\psi}_{2^C,\psi}(\mathcal{T}') = \mathcal{T}$, iff $l(d \cdot d') = l'(d \cdot d'')$ holds for all $d \in \downarrow D^\psi_\Psi \cap \downarrow D_\Psi$, $d' \in (\{\psi\} \times 2^{Scope(\psi) \smallsetminus C})^*$, $d'' \in D^*_\psi$, and $d' = hide_{2^C}(d'')$.

Let $\psi = \dashv CQs.\vartheta'$ be a minimal formula of Ψ and a direct quantified sub-formula of $\vartheta \in \Psi$, $\Psi' = \Psi \smallsetminus \{\psi\}$, and $\langle \downarrow D_{\Psi'}, l\rangle$ a Σ^s_Ψ-labeled down-tree. Then we denote with $copy_\psi \langle \downarrow D_{\Psi'}, l\rangle$ the Σ^s_Ψ-labeled down-tree $\langle \downarrow D^\psi_\Psi, l'\rangle$ with $l'(x_\psi) = l(x_\vartheta)$, where x_ϑ is derived from x_ψ by changing every letter (ψ, V) to (ϑ, V).

Theorem 4. *For a given CL specification φ, we can build an alternating automaton that accepts a 2^Π-labeled $2^{\mathcal{F}}$-tree iff it is a model of φ.*

Construction: The starting point of our construction is the automaton $\mathcal{A}^1_\Phi = \mathcal{U}^\varphi$ from Lemma 1 that accepts a Σ_Φ-labeled down-tree $\langle \downarrow D_\Phi, l\rangle$ iff it is path accepting. Following the reverse of the structure of the adjusted acceptance game, we stepwise transform $\mathcal{A}^1_\Phi$ into an automaton $\mathcal{A}^3_\emptyset$ (for quantified) or $\mathcal{A}^1_{\{\varphi\}}$ (for unquantified formulas φ) that recognizes the models of φ.

For a non-empty subset $\Psi \subseteq \Phi$ of Φ, we first choose a sub-formula $\psi \in \Psi$ such that no true sub-formula of ψ is in Ψ ($dqsf(\psi) \cap \Psi = \emptyset$), and set $\Psi' = \Psi \smallsetminus \{\psi\}$.

Let $\psi = \dashv CQs.\psi'$. Note that s occurs only in the labels of ψ-trees, and only influences their acceptance, because they are only interpreted by the automaton A^1_Ψ, iff it is in a ψ-state.

The **first transformation** refers to the $\forall$**s.** or $\exists$**s.** part of the specification, or to the fourth step of the adjusted acceptance game. Using Theorem 2, we construct an automaton $\mathcal{A}^4_{\Psi'}$ that accepts a Σ^s_Ψ-labeled down-tree $\langle \downarrow D_\Psi, l\rangle$ iff

- all Σ_Ψ-labeled down-trees $\langle \downarrow D_\Psi, l'\rangle$ with $proj_{2^{\{s\}}}(\langle \downarrow D_\Psi, l'\rangle) = \langle \downarrow D_\Psi, l\rangle$ are accepted by $\mathcal{A}^1_\Psi$ (for $Q = \forall$)), or
- some Σ_Ψ-labeled down-tree $\langle \downarrow D_\Psi, l'\rangle$ with $proj_{2^{\{s\}}}(\langle \downarrow D_\Psi, l'\rangle) = \langle \downarrow D_\Psi, l\rangle$ are accepted by $\mathcal{A}^1_\Psi$ (for $Q = \exists$)), respectively,

by applying the respective projection to the standard tree automaton that results from restricting $\mathcal{A}^1_\Psi$ to the ψ-states, with the initial ψ-state as initial state, and then replacing the old ψ-states (and transitions from there) by the new ones.

The **second transformation** refers to the $\dashv$**C** part of the specification, or to the third step of the adjusted model-checking game. We build an automaton $\mathcal{A}^3_{\Psi'}$

that accepts a Σ_Ψ^s-labeled down-tree $\langle{\downarrow}D_\Psi^\psi, l\rangle$ iff its widening $\mathit{wide}_{2^C,\psi}^{D_\psi}(\langle{\downarrow}D_\Psi^\psi, l\rangle)$ is accepted by $\mathcal{A}_{\Psi'}^4$. This transformation is the narrowing from Theorem 1 applied to the standard tree automaton that results from restricting $\mathcal{A}_{\Psi'}^4$ to the ψ-states. It only affects the transitions from the ψ-states.

The **third transformation** refers to **binding the extension to the labeled sub-tree**, or to copying the current sub-tree in the second step of the adjusted model-checking game. In the first step of this transformation, we construct the automaton ${\mathcal{A}_{\Psi'}^2}' = (\Sigma_\Psi^s, Q_\Psi, \overline{\varphi}, \delta_\Psi, \alpha_\Psi)$ with the same states as $\mathcal{A}_{\Psi'}^3$ that accepts a Σ_Ψ^s-labeled down-tree $\langle{\downarrow}D_{\Psi'}, l\rangle$ if $\mathit{copy}_\psi\langle{\downarrow}D_{\Psi'}, l\rangle$ is accepted by $\mathcal{A}_{\Psi'}^3$ by simply replacing every direction (ψ, V) by (ϑ, V), where ϑ is the formula in Ψ for which $\psi \in \mathit{dqsf}(\vartheta)$ holds: It obviously makes no difference if we continue in the blueprint or the copy. In a second step, we apply a minor change to ${\mathcal{A}_{\Psi'}^2}'$ to properly turn the ψ-states into ϑ-states. For this change, we consider that the initial ϑ-state q_0^ϑ and $\widetilde{\psi}$ are only called from the state $\overline{\vartheta}$ in the transition $\delta(\overline{\vartheta}, \sigma) = (q_0^\vartheta, \varepsilon) \wedge \bigwedge_{\psi'\sigma\cap \mathit{dqsf}(\vartheta)\cap\Psi} \delta(\widetilde{\psi'}, \varepsilon)$. We can therefore introduce a new initial ϑ-state $\widetilde{q}_0^\vartheta$ and call both states, q_0^ϑ and $\widetilde{\vartheta}$, through $\widetilde{q}_0^\vartheta$ by adjusting $\delta(\overline{\vartheta}, \sigma)$ to $(\widetilde{q}_0^\vartheta, \varepsilon) \wedge \bigwedge_{\psi'\sigma\cap \mathit{dqsf}(\vartheta)\cap\Psi'} \delta(\widetilde{\psi'}, \varepsilon)$, and choosing $\delta(\widetilde{q}_0^\vartheta, \sigma) = (q_0^\vartheta, \varepsilon) \wedge (\widetilde{\psi}, \varepsilon)$. Transforming the winning strategy for either player is trivial for both steps, and $\widetilde{\psi}$, $\overline{\psi}$, and the former ψ-states become ϑ-states.

The **fourth transformation** refers to **guessing the correctness** of ψ, or to simulating the perfect oracle of the first step of the model-checking game. We build an automaton $\mathcal{A}_{\Psi'}^1$ that accepts a $\Sigma_{\Psi'}$-labeled down-tree $\langle{\downarrow}D_{\Psi'}, l\rangle$ iff some Σ_Ψ^s-labeled down-tree $\langle{\downarrow}D_{\Psi'}, l'\rangle$ with $\mathit{proj}_{2^{\{\psi\}}}(\langle{\downarrow}D_\Psi, l'\rangle) = \langle{\downarrow}D_\Psi, l\rangle$ is accepted by $\mathcal{A}_{\Psi'}^2$ by a transformation similar to the transformation from $\mathcal{A}_\Psi^1$ to $A_{\Psi'}^4$.

These transformations are repeated until we have constructed $\mathcal{A}_\emptyset^3$ if φ is quantified, and $\mathcal{A}_{\{\varphi\}}^1$ otherwise. □

Corollary 1. *The validity, satisfiability, and model-checking problems for CL are decidable.* □

The proposed decision procedure is non-elementary, and Proposition 3 shows that this is unavoidable: The complexity of the easily encodable distributed synthesis problem [6,7,8,9] implies this for the restricted class of specifications with only existential (or only universal) quantification. However, a similar effect can be observed if we concentrate on the sub-logic without coordination variables; Disallowing coordination variables leaves us with QPTL, and hence with a tower of exponents linear in the number of quantifier alternations [18].

Beyond Coordination Logic. The power of fragments of Coordination Logic raises the question if there are natural decidable extensions. The first natural extension of CL would be to allow for an arbitrary assignment of information to strategy variables. This can, for example, be done by replacing the quantifiers $\exists CQs$ of the logic by more general quantifiers $QC \rhd s$ that assign a set of coordination variables to s. This extension would allow to introduce information in a non-ordered fashion, for example, by introducing the same information multiple times or by withdrawing information. We call the resulting logic Extended Coordination Logic. While the semantics of CL naturally extends to Extended

CL—it suffices to change the quantifiers in the last two bullet points—Extended CL is undecidable, because we can encode the realizability problem for the undecidable architecture [6,7] of Figure 1c by $\exists\{x_1\} \rhd y_1.\exists\{x_2\} \rhd y_2.\varphi$ for every system specification φ.

Proposition 6. *The fragment of Extended Coordination Logic with two strategy variables and only prenex existential (or universal) quantification is undecidable.*

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49, 672–713 (2002)
2. Chatterjee, K., Henzinger, T.A., Piterman, N.: Strategy logic. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 59–73. Springer, Heidelberg (2007)
3. Schobbens, P.Y.: Alternating-time logic with imperfect recall. In: Proc. of LCMAS 2003. Electronic Notes in Theoretical Computer Science, vol. 85, pp. 82–93 (2004)
4. Pinchinat, S.: A generic constructive solution for concurrent games with expressive constraints on strategies. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 253–267. Springer, Heidelberg (2007)
5. Brihaye, T., Lopes, A.D.C., Laroussinie, F., Markey, N.: ATL with strategy contexts and bounded memory. In: Artemov, S., Nerode, A. (eds.) LFCS 2009. LNCS, vol. 5407, pp. 92–106. Springer, Heidelberg (2008)
6. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proc. of FOCS, pp. 746–757 (1990)
7. Rosner, R.: Modular Synthesis of Reactive Systems. PhD thesis, Weizmann Institute of Science, Rehovot, Israel (1992)
8. Kupferman, O., Vardi, M.Y.: Synthesizing distributed systems. In: Proc. of LICS, pp. 389–398 (2001)
9. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proc. of LICS, pp. 321–330 (2005)
10. Madhusudan, P., Thiagarajan, P.S.: Distributed controller synthesis for local specifications. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 396–407. Springer, Heidelberg (2001)
11. Pinchinat, S., Riedweg, S.: A decidable class of problems for control under partial observation. Information Processing Letters 95, 454–460 (2005)
12. van der Hoek, W., Wooldridge, M.: Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. Studia Logica 75, 125–157 (2003)
13. Ågotnes, T., Goranko, V., Jamroga, W.: Alternating-time temporal logics with irrevocable strategies. In: Proc. of TARK, pp. 15–24 (2007)
14. Kupferman, O., Vardi, M.Y.: Synthesis with incomplete informatio. In: Proc. of ICTL (1997)
15. Schewe, S., Finkbeiner, B.: Semi-automatic distributed synthesis. International Journal of Foundations of Computer Science 18, 113–138 (2007)
16. Schewe, S., Finkbeiner, B.: Satisfiability and finite model property for the alternating-time μ-calculus. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 591–605. Springer, Heidelberg (2006)
17. Finkbeiner, B., Schewe, S.: Coordination logic. Reports of SFB/TR 14 AVACS 63. SFB/TR 14 AVACS (2010), `http://www.avacs.org`, ISSN: 1860-9821
18. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. Theoretical Computer Science 49, 217–237 (1987)