Coordinator synthesis

Suguman Bansal
Rice University
suguman@rice.edu

Kedar S. Namjoshi
Nokia Bell Labs, Murray Hill
kedar.namjoshi@nokia-bell-labs.com

Yaniv Sa'ar
Nokia Bell Labs, Kfar Saba
saar@nokia-bell-labs.com

Abstract

The design of a coordinator for multiple, independent, reactive agents is a complex task. *Coordination synthesis* is the automated construction of a coordinator from a specification and behavioral description of the reactive agents. Prior work on coordination synthesis make use of two critical assumptions: (a). all reactive agents are synchronized, (b). the coordinator has complete information about the reactive agents. However, we argue that realistic multi-agent scenarios violate both of these assumptions, rendering existing techniques unsuitable for coordination synthesis. To this end, this work presents an algorithm for coordination synthesis with *both* asynchrony and partial information. Our synthesis procedure uses high-level languages for specifications and agents, namely we use linear temporal properties for specifications and Communicating Sequential Processes (CSPs) for the reactive agents.

Keywords Coordinator, Reactive systems, asynchronous, partial information, synthesis, linear temporal logic

1 Introduction

Coordinated multi-agent systems are seeing increased adoption to achieve complex tasks. These tasks may range from maintaining ambient conditions in domestic-purpose smart buildings, where readings from sensors should be linked to heating and cooling devices, to industrial warehouses with a group of package-carrying robots that coordinate to minimize wasted effort. Typically in these settings, the individual agents are reactive and a centralized coordinator interacts with them in order to achieve the task. Since the onus of achieving the task largely lies on the the centralized coordinator, it is important to design a *provably correct coordinator*.

Coordination synthesis is the automated construction of a coordinator from a specification of the desired behavior of the fully coordinated system. Prior investigations have developed algorithms and tools for the same. Most of these make at least one of the following two critical assumptions: (a). all reactive agents are synchronized with each other and the coordinator, (b). the coordinator has complete knowledge of the local states of all of the reactive agents at all times [1, 13]. However, neither of these assumptions hold in realistic scenarios. Modern software and hardware systems harness asynchronous interactions and partial information to improve speed, responsiveness, and power consumption: delay-insensitive circuits, networks of sensors, multi-threaded programs and interacting web services are some concrete instances. In fact, asynchrony and partial information is the norm and not the exception. Hence, existing synthesis algorithms are unsuitable for coordination synthesis. To this end, this work presents a synthesis algorithm that constructs a coordinator that is correct under both asynchrony and partial information.

To drive home how naturally asynchrony and partial information appear in coordinated multi-agent systems, consider the following scenario. An industrial warehouse may consist of multiple independent, reactive robots each performing the task assigned to it such as surveillance, package-delivery, packaging and so on. These robots may interact among themselves or with a centralized coordinator to ensure smooth functioning of the warehouse. While these independent robots may synchronize with each other or the coordinator at some points (a package can be delivered only after it has been packed), some robots may not synchronize with others at all (surveillance robot does not synchronize with the others at all) and operate at their own clock. This introduces asynchrony among agents/coordinator in the warehouse. Second, the agents could change their state through private interactions that are not observed by a coordinator. In this case, the coordinator can have partial information of the local state of component robots only.

The seminal Pnueli-Rosner algorithm for asynchronous LTL synthesis [16] and its follow-ups [11] handle asynchrony as well. Here the coordinator interacts with its chaotic environment by reading from and writing to interface variables. This model was inspired by hardware, but it is at a too low level and non-intuitive for describing the agents. To this end, our algorithm works in a more natural setting where agent behavior is modeled mathematically in the framework of Communicating Sequential Processes (CSP) [10], and specifications are described by Linear-time Temporal Logic (LTL) [14] or Linear-time Temporal Logic interpreted over finite traces (LTL_f) [5]. Recent work on CSP-based synthesis [4] handles partial information but not asynchrony, and only the GR(1) fragment of LTL. The new algorithm removes both restrictions. It crucially relies on simplifications developed recently in [2] for asynchronous synthesis in the Pnueli-Rosner model, consequently solving coordination

SYNT 19, July 14, 2019, New York, NY, USA
2019.

synthesis by an efficient polynomial-time reduction to synchronous synthesis from a new (regular) specification [16].

2 Illustrative Example

Smart buildings have multiple mutually-interacting devices that are coordinated to maintain optimal conditions in the building, such as temperature, humidity, lighting, and so on. Consider, as a simple illustrative example, a *smart thermostat* that interacts with a room-temperature sensor (sensor, in short), a heater, and an air-conditioner to maintain a comfortable room temperature. The temperature is affected by the mode (switch-on or switch-off) of the heater and air-conditioner, and by external physical factors such as weather fluctuations, which are unpredictable and cannot be controlled. These external factors introduce asynchrony and partial information in the model, and prevent the smart thermostat from assessing the room temperature correctly from the modes of the devices alone. As a result, the smart thermostat must communicate with the sensor to check the room temperature, and respond accordingly.

CSP processes modeling the sensor, heater and air-conditioner are given in Figures 1-3. The states of the sensor denote its internal state, while states of the heater and air-conditioner denote their mode. The dashed-transitions in the sensor model the fluctuations in room temperature caused by changing external physical conditions and are private to the environment. The actions HeatIsOn and AcIsOn are private interactions between the sensor and the heater or air-conditioner, which model the effect that those devices have on the sensor reading. Finally, the sensor communicates the current room temperature to the smart thermostat through actions TooCold, JustRight, TooWarm, and the heater and air-conditioner interact with the smart thermostat through the Switch actions. The specification is modeled as Infinitely Often (JustRight), as the uncontrollable external temperature fluctuations make it impossible to claim that the sensor reading is always JustRight. This is easily expressible in LTL.

3 Coordination synthesis

The technical contributions of this work are two-fold.

1. We formulate the problem of coordination synthesis in presence of both asynchrony and partial information (§ 3.1). Our problem formulation uses CSPs to model reactive agents, as opposed to primitive models based on reading and writing to shared interface variables.
2. We reduce coordination synthesis to synchronous synthesis of a new regular specification. This reduction is efficient as as the new specification is linear in size of the environment and automata representing φ_S and φ_L. Therefore, it is able to leverage advances in synchronous synthesis to develop efficient tools for coordination synthesis.

For sake of brevity, we present the highlights of our contributions only.

3.1 Problem formulation

Reactive agent model

We use Communicating Sequential Processes (CSP) [10] to represent the reactive agents. A *CSP process*, or process (in short) is defined by a tuple $P = (S, \iota, \Sigma, \Gamma, \delta)$, where S is a finite set of states, $\iota \in S$ is a special start state, Σ are the publicly visible events of the process, and Γ are the privately visible events of the process. The sets Σ and Γ are disjoint. The transition relation $\delta : S \times (\Sigma \cup \Gamma) \to 2^S$ maps each state and event to a set of successor states. A transition from state s on event a to state t exists if $t \in \delta(s, a)$.

Let P and Q be CSP processes. Let X be a subset of their common public events, i.e., $X \subseteq (\Sigma_P \cap \Sigma_Q)$. The composition of P and Q relative to X, denoted $P \parallel_X Q$, is a CSP process, with state set is $S_P \times S_Q$, initial state (ι_P, ι_Q), public events $(\Sigma_P \cup \Sigma_Q) \setminus X$, private events $(\Gamma_P \cup \Gamma_Q \cup X)$, and a transition relation defined by the following rules.

- (Pairwise Synchronization) For an event a in X, there is a transition from (s, t) to (s', t') on a if (s, a, s') is a transition in P and (t, a, t') a transition in Q.
- (Internal) For an event b in Γ_P or in $\Sigma_P \setminus X$ (i.e., private, or unsynchronized public event), there is a transition from (s, t) to (s', t) on b if there is a transition (s, b, s') in P. A similar rule applies to such events in Q.

The definition forces P and Q to synchronize on events in X; for other events, the processes may act independently.

Temporal Specifications

A CSP process can have computations of two types: those that are finite, ending in dead-end states; and those that are infinite. A correctness specification should accommodate both types. Hence, we define a *correctness specification*, φ, over an action alphabet, Σ, as a pair (φ_S, φ_L), where φ_S and φ_L is a set of finite and infinite sequences over Σ and are represented in in LTL_f [5] and LTL [14], respectively.

Problem formulation

The synthesis problem is defined as follows, new terms in this definition are motivated and defined below.

Definition 3.1 (Coordination Synthesis)**.** Given an environment process $E = (S, \iota, \Sigma, \Gamma, \delta)$ and a specification φ over actions in $(\Sigma \cup \Gamma)$, construct a process M with public event set Σ such that all of the maximal finite computations of $E \parallel_\Sigma M$ satisfy φ_S and all of its infinite *fair* computations satisfy φ_L. The instance (E, φ) is *realizable* if there is a process M such that $E \parallel_\Sigma M$ has these properties.

The problem definition uses only one environment process E because we assume it represents the synchronization of all reactive agents. A process M is *non-blocking* for process

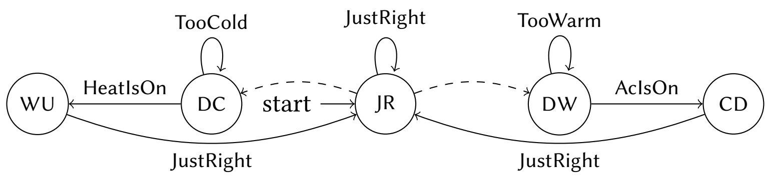

Figure 1. Room temp. sensor (JR: Just Right, DW: Detected Warm, DC: Detected Cold, WU: Warming Up, CD: Cooling Down)

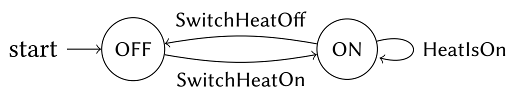

Figure 2. Heater

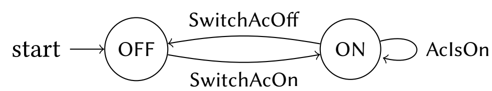

Figure 3. Air conditioner

E if all maximal computations of $E \parallel M$ are infinite. If φ_S is the empty set, any solution M must be non-blocking.

3.2 Methodology and results

This sections highlights our major results for coordination synthesis. First of all, we establish decidability of coordination synthesis via an automata-theoretic reduction of coordination synthesis to synchronous synthesis of a new (regular) specification (Theorem 3.2).

Theorem 3.2. *Given environment process E and specification $\varphi = (\varphi_S, \varphi_L)$. There exists a co-Büchi specification B such that*

- *Coordination synthesis with E and φ is realizable iff synchronous synthesis with the specification B is realizable.*
- *Solution to synchronous synthesis with B induces a CSP controller for the controller synthesis with E and φ.*
- $|B| = O(|E| \cdot |\mathcal{A}_S| \cdot |\mathcal{A}_L|)$ *where $\mathcal{A}_i$ is the automata corresponding to φ_i where i is either L or S.*

We also present its complexity-theoretic analysis.

Theorem 3.3. *Given environment process E and specification $\varphi = (\varphi_S, \varphi_L)$. Coordination synthesis is 2EXPTIME-complete in $|\varphi|$, and PSPACE-hard in $|E|$.*

This is an encouraging theoretical result since even the simple synchronous synthesis with temporal specification is 2EXPTIME-complete in the specification. However, the hardness in size of E could be a source of blowup in practice and a problem to be resolved in future work. Finally, the addition of fairness constraints to the controller is an important direction for future research.

4 Related work

Synthesis of synchronous reactive systems The synthesis question for temporal properties originates from a question posed by Church in the 1950s (see [19]). The problem of synthesizing a synchronous reactive system from a linear temporal specification was formulated and studied by Pnueli and Rosner [15], and can be generalized to regular specifications. Much progress on the synchronous synthesis question has lead to efficient techniques [9, 12, 18] and scalable tools, e.g. [3, 6–8, 17]. The new specification constructed in Theorem 3.2 is passed into one of these tools to solve coordination synthesis.

Acknowledgements. Kedar Namjoshi and Suguman Bansal were supported, in part, by NSF grant CCF-1563393.

References

[1] R. Alur, S. Moarref, and U. Topcu. 2016. Compositional synthesis of reactive controllers for multi-agent systems. In *Proc. of CAV*. 251–269.
[2] S. Bansal, K. S. Namjoshi, and Y. Saar. 2018. Synthesis of Asynchronous Reactive Programs from Temporal Specifications. In *Proc. of CAV*.
[3] A. Bohy, V. Bruyère, E. Filiot, N. Jin, and J. F. Raskin. 2012. Acacia+, a Tool for LTL Synthesis.. In *Proc. of CAV*.
[4] D. Ciolek, V. A. Braberman, N. D'Ippolito, N. Piterman, and S. Uchitel. 2017. Interaction Models and Automated Control under Partial Observable Environments. *IEEE Trans. Software Eng.* 43, 1 (2017).
[5] G. De Giacomo and M. Y. Vardi. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. of AAAI*.
[6] R. Ehlers. 2010. Symbolic Bounded Synthesis.. In *Proc. of CAV*.
[7] R. Ehlers. 2011. Unbeast: Symbolic bounded synthesis. In *Proc. of TACAS*.
[8] P. Faymonville, B. Finkbeiner, and L. Tentrup. 2017. BoSy: An Experimentation Framework for Bounded Synthesis. In *Proc. of CAV*.
[9] E Filiot, N Jin, and J. F. Raskin. [n. d.]. Compositional Algorithms for LTL Synthesis. In *Proc. of ATVA*.
[10] C. A. R. Hoare. 1978. Communicating Sequential Processes. *Commun. ACM* 21, 8 (1978).
[11] U. Klein, N. Piterman, and A. Pnueli. 2012. Effective synthesis of asynchronous systems from GR (1) specifications. In *VMCAI*.
[12] O. Kupferman and M. Y. Vardi. 2005. Safraless decision procedures. In *Proc. of FOCS*.
[13] S. Moarref and H. Kress-Gazit. 2018. Reactive Synthesis for Robotic Swarms. *Formal Modeling and Analysis of Timed Systems*, 71–87.
[14] A. Pnueli. 1977. The temporal logic of programs. In *Proc. of FOCS*.
[15] A. Pnueli and R. Rosner. 1989. On the synthesis of a reactive module. In *POPL*.
[16] A. Pnueli and R. Rosner. 1989. On the synthesis of an asynchronous reactive module. *In Proc. of ICALP* (1989).
[17] A. Pnueli, Y. Saar, and L. D. Zuck. 2010. JTLV: A Framework for Developing Verification Algorithms. In *Proc. of CAV*.
[18] S. Schewe and B. Finkbeiner. 2007. Bounded synthesis. (2007).
[19] W. Thomas. 2009. Facets of Synthesis: Revisiting Church's Problem. In *Proc. of FOSSACS*.